STOCHASTIC OPTIMIZATION AND LEARNING

A unified framework (Draft)

Warren B. Powell

March 21, 2019

WILEY-
INTERSCIENCE

A JOHN WILEY & SONS, INC., PUBLICATION

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created ore extended by sales representatives or written sales materials. The advice and strategies contained herin may not be suitable for your situation. You should consult with a professional where appropriate. Neither the publisher nor author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

For general information on our other products and services please contact our Customer Care Department with the U.S. at 877-762-2974, outside the U.S. at 317-572-3993 or fax 317-572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print, however, may not be available in electronic format.

Library of Congress Cataloging-in-Publication Data:

Optimization Under Uncertainty: A unified framework
Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

CONTENTS

PART I - INTRODUCTION

We begin our journey by providing an overview of the diversity of optimization problems under uncertainty. These have been introduced over the years by a number of different communities, motivated by different applications. Many of these problems have motivated entire fields of research under names such as dynamic programming, stochastic programming, optimal control, stochastic search, ranking and selection, and multiarmed bandit problems.

CHAPTER 1

DECISIONS AND UNCERTAINTY

$$\rho\{...\}, \mathbb{E}\{...\}$$

There are few problems that offer the richness and diversity of making decisions in the presence of uncertainty. Decision making under uncertainty is a universal experience, something every human has had to manage since our first experiments trying new foods when we were two years old. Some samples of everyday problems where we have to manage uncertainty include:

- Personal decisions - These might include how much to withdraw from an ATM machine, finding the best path to a new job, and deciding what time to leave to make an important appointment.
- Health decisions - Examples include joining a health club, getting annual checkups, having that mole checked, using dental floss, and scheduling a colonoscopy.
- Investment decisions - What mutual fund should you use? How should you allocate your investments? How much should you put away for retirement? Should you rent or purchase a house?
- Professional decisions - These are the decisions we have to make as part of our jobs, or whether to take a job.

Decisions under uncertainty span virtually every major field. Samples include

Introduction to Stochastic Programming
Handbook of Simulation Optimization
Robust Optimization
Introduction to Decision Analysis
Approximate Dynamic Programming
Optimal Learning
Model Predictive Control
Dynamic Programming and Optimal Control
Introduction to Stochastic Search and Optimization
Multi-Armed Bandit Allocation Indices
Optimal Control
Reinforcement Learning
Markov Decision Processes
Online Computation and Competitive Analysis
Stochastic Simulation Optimization
Stochastic Controls

Figure 1.1 A sampling of major books representing different fields in stochastic optimization.

- Business - What products to sell, with what features? Which supplies should you use? What price should you charge? How should we manage our fleet of delivery vehicles? Which menu attracts the most customers?
- Internet - What ads to display to maximize ad-clicks? Which movies attract the most attention? When/how should mass notices be sent?
- Engineering - How to design devices from aerosol cans to an electric vehicle, bridges to transportation systems, transistors to computers?
- Materials science - What combination of temperatures, pressures and concentrations should we use to create a material with the highest strength?
- Medical research - What molecular configuration will produce the drug which kills the most cancer cells? What set of steps are required to produce single-walled nanotubes?
- Economics - What interest rate should the Federal Reserve charge? What levels of market liquidity should be provided? What guidelines should be imposed on investment banks?

Needless to say, listing every possible type of decision is an impossible task. However, we would argue that in the context of a particular problem, listing the decisions is easier than identifying all the sources of uncertainty.

There has been a historical pattern to pick up the modeling styles and solution approaches used in the different books captured in figure 1.1. These fields include:

- Decision analysis - This community generally works with discrete actions, possibly discrete random outcomes, but often features complex utility functions and the handling of risk. Problems are relatively small.
- Stochastic search (derivative based) - This field is centered on the basic problem $\min_x \mathbb{E}F(x,W)$ where x is continuous (scalar or vector), W is a random variable,

and where the expectation typically cannot be computed. However, we assume we can compute gradients $\nabla_x F(x, W)$ for a known W.

- Ranking and selection (derivative free) - This field is also focused on $\min_x \mathbb{E}F(x, W)$, but now we assume that x can take on one of a finite set of outcomes $\{x_1, x_2, \ldots, x_M\}$.

- Simulation-optimization - This community evolved from within the setting of discrete event simulations, where we need to use a simulator (such as one of a manufacturing system) to compare different designs. The field of simulation optimization started with the ranking and selection problem, but has evolved to span a wider range of problems.

- Online computation - This field describes methods where decisions are made which simply react to information as it comes in, without considering the impact of decisions now on the future. This field was originally motivated by mobile applications where energy and computing capacity was limited.

- Optimal control - The roots of this community are in engineering, focusing on the control of aircraft, spacecraft, and robots, but has expanded to economics and computer science. The original problems were written in continuous time with continuous controls, but is often written in discrete time (typically with discrete controls), since this is how computation is done. Problems are typically deterministic, possibly with uncertain parameters, and possibly with additive noise in the transition function, but this community has been widely adopted, especially in finance where problems are purely stochastic.

- Robust optimization - This is an extension of stochastic search with roots in engineering, where the goal is to find the best design x (of a building, an aircraft, a car) that works under the worst instance of a random variable W (which could be wind, temperature, crash conditions). Instead of $\min_x \mathbb{E}F(x, W)$, robust optimization problems seek to solve $\min_x \max_w F(x, w)$. For example, we might want to design a wing to handle the worst possible wind stresses.

- Optimal stopping - This is an important problem in finance, where we have to study when to stop and sell (or buy) an asset. It also arises in engineering when we have to decide when to stop and repair a piece of machinery. The problem is to find a time τ to sell or repair, where τ can only depend on the information available at that time. The problem is popular within the applied probability community.

- Markov decision processes - This community evolved primarily within applied probability, and describes a system that takes on a discrete state s, and transitions to s' when we take (discrete) action a with (known) probability $p(s'|s, a)$.

- Reinforcement learning - This field started by modeling animal behavior seeking to solve a problem (such as finding a path through a maze), where experiences (in the form of successes or failures) were captured by estimating the value of a state-action pair.

- Approximate dynamic programming - Several communities have developed ways of overcoming the “curse of dimensionality” inherent in the tools of discrete Markov decision processes by using simulation-based methods to solve dynamic programs.

There are very close parallels between the fields of approximate dynamic programming (also known as adaptive dynamic programming) and reinforcement learning, although the motivating problems tend to be somewhat different, as are the research styles.

- Stochastic programming - This community evolved from math programming with the desire to insert random variables into linear programs. The classical problem is the two-stage problem where you pick x_0 (e.g. how many Christmas trees to plant), after which we learn random information W_1, and then we make a second decision x_1 (e.g. shipping Christmas trees to customers).
- Sequential kriging - This community evolved within geosciences, where we need to learn the largest value of a continuous function $f(x)$ through expensive experiments (originally field experiments). The vector x was originally a two-dimensional point in space, where $f(x)$ might be the amount of oil or natural gas yield by drilling a hole at x.
- Multiarmed bandit problems - The roots of this problem come from applied probability, where the goal is to identify a discrete alternative (known as an "arm" in this community) that yields the highest reward, where we do not know the value of each "arm." We learn the value through repeated experimentation, accumulating rewards as we progress.

This list provides a sense of the different communities that have been drawn into the arena of making decisions under uncertainty. Some of these communities have their roots in deterministic problems (optimal control, stochastic programming), while others have their roots in communities such as applied probability and simulation. A byproduct of this confluence of communities is a variety of notational systems and mathematical styles.

This diversity of languages disguises common approaches developed in different communities. Less important than this process of re-invention of methods is the potential for cross-fertilization of ideas across communities. Indeed, as of this writing there persists a fair amount of competition between communities where proponents of one methodological approach will insist that their approach is better than another.

We organize all of these fields under the broad term "stochastic optimization and learning" which captures not just the communities listed above that deal with decisions under uncertainty, but also the fields that address the many learning problems that arise in most (but not all) solution approaches. In addition, we also have to recognize the importance of developing stochastic models of the underlying problem. Thus, stochastic optimization requires integrating three core communities from the mathematical sciences:

Mathematical programming This field covers the core methodologies in derivative-based and derivative-free search algorithms, including the broad fields of linear, nonlinear and integer programming. We will draw on the tools of math programming in settings to find the best decision, or to find the parameters that produce the best model.

Statistical learning Here we bring together the fields of statistics, machine learning and data sciences. Most (although not all) of our applications of these tools will involve recursive learning. We will also draw on the fields of both frequentist and Bayesian statistics.

Stochastic modeling Optimizing a problem in the presence of uncertainty often requires a careful model of the uncertain quantities that affect the performance of a process. Stochastic modeling draws tools from the fields of probability, simulation, and uncertainty quantification.

This book is not intended to replace the much more thorough treatments of the more specialized books that focus on specific modeling approaches and algorithmic strategies. Rather, our goal is to provide a unified framework that provides a more comprehensive perspective of these fields. We have found that a single problem can be reasonably approached by techniques from multiple fields such as dynamic programming (operations research), model predictive control (control theory) and policy search (computer science), where any one of these methods may work best, depending on the specific characteristics of the data. At the same time, powerful hybrid strategies can be created by combining the tools from different fields.

1.1 WHY A UNIVERSAL FORMULATION?

The diversity of problems is so broad that one might ask: can all these problems be covered in a single book? And even if this is possible, what is the benefit?

The problem that practitioners find when solving real problems is that the computational tools of the different communities are *fragile*. By this we mean that they break down quickly with surprisingly small changes in the characteristics of a problem. Some examples of this include:

- The field of decision analysis uses the powerful idea of decision trees which involves enumerating decisions and uncertainties. While the computational demands depend on the problem, this approach might work over a horizon of, say, five time periods, and break down entirely if we try to extend to six or seven time periods.

- The field of discrete Markov decision processes can typically handle state variables with one or two dimensions with run times growing from seconds to minutes to an hour or more. Going to three dimensions can push run times into the range of days to several weeks. Four dimensions is typically impossible except in special cases. Real problems range from a few dimensions to many thousands or millions of dimensions.

- A classical stochastic optimization problem involves deciding the best quantity x to meet a demand D, where we sell the smaller of x and D at a price p while purchasing x at a cost c. Known as the *newsvendor problem*, this is written as

  $$\max_x \mathbb{E}\{p\min(x,D) - cx\}$$

  where the distribution of demand D is a random variable with an unknown distribution (which means we cannot compute the expectation). The multiarmed bandit literature proposes rules (policies) for testing different values of x to maximize an unknown function.

  Now assume that before choosing the quantity x_t at time t, we are given a price p_t, which means we are now solving

  $$\max_x \mathbb{E}\{p_t\min(x,D) - cx\}.$$

  The price p_t changes each time we need to make another decision x_t. This small change produces a fundamentally different problem which requires entirely different algorithmic strategies.

- The field known as stochastic programming provides methods for solving stochastic resource allocation problems under uncertainties such as random demands and prices. However, the algorithmic strategy breaks down completely if the random demands and prices depend on decisions (for example, selling a large quantity might depress prices).

- The optimal control community focuses on solving the deterministic problem

$$\min_{u_0,\ldots,u_T} \sum_{t=0}^{T} \big((x_t)^T Q_t x_t + (u_t)^T R_t u_t\big)$$

  where the state x_t (which might be the location and velocity of a robot), acted on with a control u_t (a force), evolves according to $x_{t+1} = f(x_t, u_t)$. The optimal solution has the form $U^*(x_t) = K_t x_t$ where the matrix K_t depends on Q_t and R_t. Note that the controls are unconstrained. Introduce a single constraint such as $u_t \geq 0$ and this solution is lost.

- The multiarmed bandit problem involves finding an alternative (e.g. an online ad) that produces the greatest number of clicks (that may result in sales of a product). The problem is to find the ad that produces the greatest number of clicks for a particular user. A famous result known as Gittins indices gives an optimal policy by choosing the ad with the greatest index. This result requires, however, that we are optimizing the discounted number of ad-clicks over an infinite horizon, and that the underlying market behavior does not change (both of these conditions are typically violated in most problem settings).

- Reinforcement learning can be used to optimize the movement of a single taxi driver, but the methods do not extend to handling two taxis (at the same time). Fleets in a city might have hundreds or even thousands of vehicles.

- Approximate dynamic programming has been found to work exceptionally well optimizing the storage of energy between 500 batteries over a grid if the only sources of uncertainty are random supplies (e.g. from wind) and demands (which depends on temperature). However, it struggles with even one battery if we try to model just one or two exogenous information processes such as weather or prices.

- Inventory problems I: Consider an inventory problem where we sell buy and sell a quantity x_t from our inventory R_t at a fixed price p. For this problem the state S_t is just the inventory R_t, and the problem can typically be solved in a few minutes. If we sell at a stochastically varying price p_t (where p_t is independent of p_{t-1}), then the state is $S_t = (R_t, p_t)$, and the run times grow to hours. If the price evolves according to the time series $p_{t+1} = \theta_0 p_t + \theta_1 p_{t-1} + \varepsilon_{t+1}$, then the state is $S_t = (R_t, p_t, p_{t-1})$. Now the solution of Bellman's equation could take a week or more.

- Inventory problems II: Now imagine that our inventory is time-varying but that we have a forecast of demand given by $f^D_{tt'}$. We would use fundamentally different types of solution approaches if the forecast is reasonably accurate, or very inaccurate.

In short, after learning a particular modeling and algorithmic framework, surprisingly minor details can "break" an algorithmic approach.

Reflecting this state of affairs, different fields of stochastic optimization have grown (and continue to grow as this is being written) into neighboring areas as the respective research communities tackle new challenges for which the original techniques no longer work. As a result, communities with names such as "stochastic search" (the oldest community), "reinforcement learning," "simulation optimization," and "multiarmed bandit problems," all with their roots in very distinct problems, are steadily morphing as they discover ideas that have been at least partly developed by their sister communities in stochastic optimization.

This book overcomes this problem by accomplishing the following:

A universal formulation - The entire range of problems suggested above can be modeled, with perhaps a few adjustments, using a single formulation. This formulation is quite general, and makes it possible to model a truly diverse range of problems with a high level of realism.

Cross fertilization - Ideas developed from one problem class or discipline can be used to help solve problems traditionally associated with different areas.

Thus, while we do not offer a magic wand that will solve all problems (this problem class is too diverse), our approach brings together the skills and perspectives of all of the communities. Recognizing how to bring all these fields together requires, of course, a common modeling framework, which we provide.

We then draw on all the communities that have contributed to this broad problem class to identify two fundamental strategies for designing policies (also known as decision rules) for solving these problems. These two strategies each lead to two classes of policies, producing four fundamental classes of policies which we use as a foundation for solving all problems. Small changes to problems can lead us to move from one policy class to another, or to build hybrids. However, the four classes of policies will form the foundation of all of our solution approaches.

In the process, we will see that bringing the perspective of one community (say, multiarmed bandit problems) to others (such as gradient-based stochastic search or dynamic programming) helps to identify not just new tools to solve existing problems, but also opens up entirely new research questions.

1.2 SOME SAMPLE PROBLEMS

A few sample problems provides a hint into the major classes of decision problems that involve uncertainty.

The newsvendor problem The newsvendor problem is one of the most widely used examples of stochastic optimization problems. Imagine that you have to decide on a quantity x of newspapers to purchase at a unit price c that will be placed in the bin the next day. You then sell up to a random demand D, charging a price p. Your profit $F(x, D)$ is given by

$$F(x, D) = p \min\{x, D\} - cx.$$

We do not know the demand D when we decide x, so we have to find x that maximizes the expected profits, given by

$$\max_x \mathbb{E}\{p \min\{x, D\} - cx\}. \tag{1.1}$$

The newsvendor problem comes in many variations, which explains its continued popularity after decades of study. One of the most important variations depends on whether the distribution D is known (which allows us to solve (1.1) analytically) or unknown (or not easily computable). Prices and costs may be random (we may be purchasing energy from the power grid at highly random prices, storing it in a battery to be used later).

Newsvendor problems can be formulated in two ways.

Offline (terminal reward) formulation - The most classical formulation, stated in (1.1), is one where we find a single solution x^* which, when implemented, solves (1.1). In this setting, we are allowed to search for the best solution without worrying about how we arrived at x^*, which means we are only interested in the *terminal reward* (that is, the quality of the solution after we have finished our search). We can further divide this problem into two formulations:

- The asymptotic formulation - This is the formulation in (1.1) - we are looking for a single, deterministic solution x^* to solve (1.1).
- The finite time formulation - Imagine that we are allowed N samples of the function $F(x, D)$ to find a solution $\bar{x}^N$, which will depend on both how we have gone about finding our solution, as well as the noisy observations we made along the way.

Online (cumulative reward) formulation - Now imagine that we do not know the distribution of the demand D, but rather have to experiment by choosing x and then observing D, after which we can figure out how much money we made that day. The problem is that we have to maximize profits over the days while we are learning the best solution, which means we have to live with the profits while we are learning the right solution.

A stochastic shortest path problem Imagine that we are trying to find the best path over an urban network. As a result of congestion, travel times on each link (i, j) joining nodes i and j may be random. We may assume that we know the cost from i to j as soon as we arrive at node i. We may assume the distribution of the cost c_{ij} is known, or unknown. In addition it may be stationary (does not change with time) or nonstationary (reflecting either predictable congestion patterns, or random accidents).

Optimizing medical decisions During a routine medical exam, a physician realizes that the patient has high blood sugar. Courses of treatment can include diet and exercise, or a popular drug called metformin that is known to reduce blood sugar. Information collected as part of the medical history can be used to guide this decision, since not every patient can handle metformin. The doctor will have to learn how this patient responds to a particular course of treatment, which is information that he can use to help guide his treatment of not only this patient, but others with similar characteristics.

Pricing an electricity contract A utility has been asked to price a contract to sell electricity over a 5-year horizon (which is quite long). The utility can exploit 5-year contracts on fuels (coal, oil, natural gas), which provide a forecast (by financial markets) on the price of the fuels in the future. Fuel prices can be translated to the cost of producing electricity from different generators, each of which has a “heat rate” that translates energy input (typically measured in millions of BTUs) and electricity output (measured in megawatt-hours). We can predict the price of electricity by

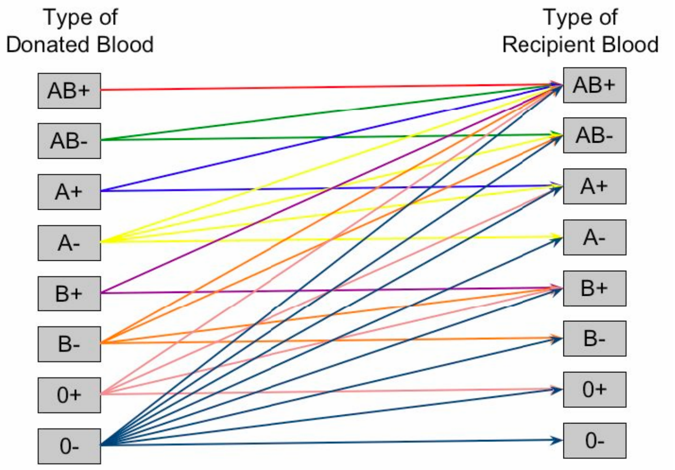


Figure 1.2 The different substitution possibilities between donated blood and patient types (from Cant (2006)).

finding the intersection between the supply curve, constructed by sorting generators from lowest to highest cost, and the projected demand. Five years from now, there is uncertainty in both the prices of different fuels, as well as the demand for electricity (known as the "load" in the power community).

Inverted pendulum problem This is one of the most classical problems in engineering control. Imagine you have a vertical rod hinged at the bottom on a cart that can move left and right. The rod tends to fall in whatever direction it is leaning, but this can be countered by moving the cart in the same direction to push the rod back up. The challenge is to control the cart in a way that the rod remains upright, ideally minimizing the energy to maintain the rod in its vertical position. These problems are typically low-dimensional (this problem has a one- or two-dimensional controller, depending on how the dynamics are modeled), and deterministic, although uncertainty can be introduced in the transition (for example to reflect wind) or in the implementation of the control.

Managing blood inventories There are eight blood types (A, B, AB, O, which can each be either positive or negative). Blood can also be stored for up to six weeks, and it may be frozen so that it can be held for longer periods of time. Each blood type has different substitution options (see figure 1.2). For example, anyone can accept O-negative blood (known as the universal donor), while A-positive blood can only be used for people with A-positive or AB-negative (known as the universal recipient). As a result of different substitution options, it is not necessarily the case that you want to use, say, A-positive blood for an A-positive patient, who can also be handled with either O-negative, O-positive, A-negative as well as A-positive blood.

Hospitals (or the Red Cross) have to periodically run blood drives, which produce an uncertain response. At the same time, demand for blood comes from a mixture of routine, scheduled surgeries and bursts from large accidents, storms and domestic violence.

If the problem is modeled in weekly time increments, blood may have an age from 0 to 5 weeks. These six values times the eight blood types, times two (frozen or not) gives us 96 values for the blood attribute. There are hundreds of possible assignments of these blood "types" to the different patient types.

These problems are but a tiny sample of the wide range of problems we may encounter that combine decisions and uncertainty. These applications illustrate both offline (design first) and online (decisions have to be made and experienced over time) settings. Decisions may be scalar, vectors (the blood), or categorical (the medical decisions). Uncertainty can be introduced in different forms. And finally, there are different types of objectives, including a desire to do well on average (the newsvendor problem is typically repeated many times), as well as to handle risk (of an outage of power or a blood shortage).

1.3 DIMENSIONS OF A STOCHASTIC OPTIMIZATION PROBLEM

Although we revisit this later in more detail, it is useful to have a sense of the different types of problems classes. We provide this by running down the different dimensions of a sequential decision problem, identifying the varieties of each of the dimensions. We wait until chapter 9 before providing a much more comprehensive presentation of how to model this rich problem class.

1.3.1 State variables

The state variable S_t captures all the information available at time t (we may also use S^n to describe the state at iteration n) that we need to model the system from time t onward. State variables come in different forms. While the labeling of different elements of the state variable is not critical, there are many problems where it is natural to identify up to three flavors of state variables:

Physical state R_t - This might be inventory, the location of a vehicle on a network, or the amount of money invested in a stock, where R_t may be a scalar (money in the bank), a low-dimensional vector (the inventory of different blood types) or a very high-dimensional vector (the number of different types of aircraft described by a vector of attributes). Physical states restrict the decisions we can make in some way. For example, we cannot sell more shares of a stock than we own. The location on a network determines the decisions we can make. In the vast majority of problems, decisions affect the physical state, directly or indirectly.

Informational state I_t - This includes information that affects the behavior of a problem, such as the temperature (that influences evaporation from a reservoir), economic variables (that influence stock prices and interest rates), medical factors (e.g. whether someone needing a knee replacement is also a smoker). An informational state variable is any relevant piece of information that we know perfectly, and that we have not classified as a physical state.

Belief state B_t - The belief state (sometimes called the knowledge state) is the information that specifies a probability distribution describing some unknown parameter. The parameter might be an unknown response to medication, the revenue produced by selling a product at a particular price, or the number of people with a medical condition (which can only be observed imperfectly).

Regardless of whether the elements of the state variable are describing physical resources such as inventories or the location of an aircraft, information such as weather or stock prices, or uncertain parameters such as the wear and tear of an engine, the condition of a patient, or how a market might respond to incentives, the state variable is information. Further, the state S_t is all the information (and only the information) needed to model the system from time t onward.

State variables may be perfectly controllable (as is often the case with physical states), uncontrollable (such as the weather), or partially controllable (such as the health of a patient being treated or the spread of treatable diseases).

1.3.2 Types of decisions

Decisions come in many different styles, and this has produced a variety of notational systems. The most common canonical notational systems for decisions are:

Discrete action a - This notation is typically used when a is discrete (binary, integer, categorical). This is widely used in computer science, which inherited the notation from the Markov decision process community in operations research.

Continuous control u - In the controls community, u is typically a low-dimensional continuous vector (say 1-10 dimensions).

General vectors x - In operations research, x is typically a vector of continuous or discrete (integer) variables, where it is not unusual to solve problems with tens or hundreds of thousands of variables (dimensions).

Regardless of the community, decisions may come in many forms. We will use our default notation of x, where $\mathcal{X}$ is the set of possible values of x.

- Binary - $\mathcal{X} = \{0, 1\}$. Binary choices arise frequently in finance (hold or sell an asset), and internet applications where $x = 0$ means "run the current website" while $x = 1$ means "run the redesigned website" (this is known as A/B testing).
- Discrete - $\mathcal{X} = \{1, 2, \ldots, M\}$.
- Subset - x may be a vector $(0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1)$ indicating, for example, the starting lineup of a basketball team.
- Scalar continuous - $\mathcal{X} = (a, b)$ for some $b > a$. This is typically written $\mathcal{X} = \mathbb{R}$.
- Continuous vector - $x = (x_1, \ldots, x_n)$ might be an n-dimensional vector, where we may write $\mathcal{X} = \mathbb{R}^n$.
- Discrete vector - x can be a vector of binary elements, or a vector of integers (0, 1, 2, ...).
- Categorical - x may be a category such as a type of drug, or a choice of employee described by a long list of attributes, or a choice of a movie (also described by a long list of attributes). If we let $a_1, a_2, \ldots, a_K$ be the different attributes of a choice, we see that the number of possible categories can be extremely large.

The nature of the decision variable, along with properties of the objective function and the nature of different forms of uncertainty, can have a major impact on the design of solution strategies.

1.3.3 Types of uncertainty

The various communities working in stochastic optimization often focus on making the best decision in the presence of some sort of uncertainty. In fact, the modeling of uncertainty is quite important. The dimensions of uncertainty modeling are

Sources of uncertainty There are over 10 different ways in which uncertainty can enter a model such as observational uncertainty, forecasting, model uncertainty, and uncertainty in the implementation of decisions.

Distributions When we use probability distributions, we can draw on a substantial library of distributions such as normal, exponential, Poisson and uniform (to name but a few).

Distribution-based vs. distribution-free We may have a formal stochastic model of uncertainty, but we often encounter problems where we do not have a probability model, and depend instead on exogenous sources of information.

1.3.4 Models of system dynamics

Almost all of the problems we consider in this volume can be modeled sequentially, starting with some state which is then modified by a decision and then new information, which then leads to a new state. However, there are different ways that we can compute, measure or observe these transitions. These include

Model-based - Here we assume we can model the transition as a system of equations. For example, an inventory problem might include the equation

$$R_{t+1} = R_t + x_t + \hat{R}_{t+1},$$

where R_t is our current inventory (say, water in a reservoir or product on a store shelf), x_t may represent purchases or (for water) releases, and $\hat{R}_{t+1}$ is random, exogenous changes (rainfall or leakage, sales or returns).

Model-free - This describes situations where we can observe a state S_t, then take an action a_t, but after that all we can do is observe the next state S_{t+1}. This might arise whenever we have a complex process such as global climate change, the dynamics of a complex production plant, a complex computer simulation, or the behavior of a human being.

Both of these are very important problem classes, and there are specialized modeling and algorithmic strategies designed for each of them.

1.3.5 Objectives

There are many ways to evaluate the performance of a system, which may involve a single metric, or multiple goals or objectives. Different classes of metrics include

Costs or contributions - These include financial metrics that we are minimizing (costs) or maximizing (profits, revenue, contributions).

Performance metrics - Here we would include non-financial metrics such as strength of a material, cancer cells killed, post-operative recovery from a surgical procedure.

Faults - We may wish to maximize a performance metric (or minimize cost), but we have to monitor service failures, or flaws in a product.

Time - We may be trying to minimize the time required to complete a task.

We also need to consider how costs (or rewards/contributions) are being accumulated:

Cumulative rewards - We may accumulate rewards (or costs) as we progress, as would happen in many online applications where we have to actually experience the process (e.g. testing different prices and observing revenues, trying out different paths through a network, or making medical decisions for patients).

Terminal rewards - We may be able to run a series of tests (in a lab, or using a computer simulation) looking to identify the best design, and then evaluate the performance based on how well the design works, and without regard to poor results that arise while searching for the best design.

We next have to consider the different ways of evaluating these metrics in the presence of uncertainty. These include

Expectations - This is the most widely used strategy, which involves averaging across the different outcomes.

Risk measures - This includes a variety of metrics that capture the variability of our performance metric. Examples include:

- Variance of the metric.
- A quantile (e.g. maximizing the 10th percentile of profits).
- Probability of being above, say, the 90th quantile or below the 10th quantile.
- Expected value given it is above or below some threshold.

Worst case - Often associated with *robust optimization*, we may wish to focus on the worst possible outcome, as might arise in the design of an engineering part where we want the lowest cost to handle the worst case. This is technically a form of risk measure, but robust optimization is a distinct field.

1.3.6 Staging of information and decisions

Stochastic problems come in a variety of flavors in terms of the sequencing of information and decisions. At one extreme is a simple "decision-information" where we make a decision, then observe some information that we did not know when we made the decision. At the other extreme is an infinite sequence of "decision-information-decision-information-..."

1.4 FORMULATING A STOCHASTIC OPTIMIZATION PROBLEM

It is useful to pause at this point and provide a hint of how we go about formulating and solving a stochastic optimization problem. Below, we use a simple inventory problem, and begin by contrasting how we formulate deterministic and stochastic versions of the problem. This requires introducing the notion of a *policy* (the controls community would call this a *control law*), which is a method for making decisions. We then briefly introduce the notion of policies and hint at how we are going to go about creating policies.

1.4.1 A deterministic inventory problem

Imagine that we want to solve a simple inventory problem, where we have to decide how much to order, x^o_t, at time t. We are going to assume that when we order x^o_t, the items cannot be used until time $t+1$. Let c_t be the cost of items ordered in period t (which can vary from one time period to the next), and assume that we are paid a price p_t when we satisfy the demand given by D_t. Let x^s_t be the sales at time t, which is limited by the demand D_t, and the available product which is our inventory R_t plus our orders x^o_t, so we can write

$$\begin{aligned} x^s_t &\leq D_t, && (1.2)\\ x^s_t &\leq R_t, && (1.3)\\ x^s_t &\geq 0, && (1.4)\\ x^o_t &\geq 0. && (1.5) \end{aligned}$$

We assume that unsatisfied demand is lost. The left-over inventory is

$$R_{t+1} = R_t + x^o_t - x^s_t. \tag{1.6}$$

Finally, we are going to let $x_t = (x^o_t, x^s_t)$.

Now we formulate our optimization problem as

$$\max_{x_t, t=0,\ldots,T} \sum_{t=0}^{T} \big(p_t x^s_t - c_t x^o_t\big), \tag{1.7}$$

subject to the constraints (1.2) - (1.6). The solution is in the form of the vector of production and sales decisions $x_t, t = 0, \ldots, T$.

It should not come as a big surprise that we need to solve the inventory problem over the entire horizon to make the best decision now. For example, if we specified in advance $x_1, \ldots, x_T$, then this could easily change what we do now, x_0, at time 0. We revisit this observation after we discuss how to handle the stochastic version of the problem.

1.4.2 The transition to a stochastic formulation

Now consider what happens when we make the demands D_t random. Since D_t is random, the inventory R_t is random. This means that the order quantity x^o_t is random, as is the sales x^s_t. Given this, the optimization problem in (1.7) simply does not make any sense (what does it mean to optimize over a set of random variables?).

We fix this by replacing the decisions $x_t = (x^o_t, x^s_t)$ with a function, known as a policy (or control law), that we designate $X^\pi(S_t) = (X^{\pi,o}_t(S_t), X^{\pi,s}_t(S_t))$ where S_t is our "state" variable that captures what we know (in particular, what we need to know) to make a decision. For this simple problem, our state variable is just the inventory R_t. This allows us to rewrite our objective function (1.7) as

$$\max_{\pi} \mathbb{E} \sum_{t=0}^{T} \big(p_t X^{\pi,s}_t(R_t) - c_t X^{\pi,o}_t(R_t)\big). \tag{1.8}$$

This equation is still solved given the constraints (1.2)-(1.6), but they are applied differently. The policy $X^\pi_t(R_t)$ has to obey equations (1.2)-(1.5), which is easy to do because at time

t, both R_t and D_t are known. Then, after computing $(x^o_t, x^e_t) = X^\pi_t(R_t)$, we can compute R_{t+1} from equation (1.6), which means sampling D_{t+1} from some distribution. Finally, the expectation $\mathbb{E}$ in (1.8) means sampling over all the different sequences $D_1, \ldots, D_T$. In practice, we cannot actually do this, so imagine that we create a sample $\hat{\Omega}$ where $\omega \in \hat{\Omega}$ represents a particular sequence of possible values of $D_1(\omega), \ldots, D_T(\omega)$. If we assume that each sequence of demands is equally likely, we would approximate (1.8) using

$$\max_\pi \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T} \big(p_t X^{\pi,s}_t(R_t(\omega^n)) - c_t X^{\pi,o}_t(R_t(\omega^n))\big). \tag{1.9}$$

where N is the number of sample paths in $\hat{\Omega}$.

Equation (1.8), which we approximate using (1.9), illustrates how easily we can transition from deterministic to stochastic optimization models. There is only one hitch: how in the world do we find these “policies”?

1.4.3 Choosing inventory policies

Before we progress too far, we have to first establish: what, precisely, is a “policy”? In a nutshell, a policy is a function that returns a feasible decision x_t (or a_t or u_t or whatever notation you are using) using the information in our state variable S_t. We have yet to formally define state variables, but for our purposes, these include any information we need to make a decision (they have to include some other information, but we return to this topic in depth later).

Policies come in different forms. For our inventory problem, we might use a simple rule such as: if the inventory R_t goes below θ^{min}, order $x^o_t = \theta^{max} - R_t$, which brings our inventory up to θ^{max}. Let $\theta^{Ord} = (\theta^{min}, \theta^{min})$. We write the policy as

$$X^{Ord}(S_t|\theta^{Ord}) = \begin{cases} \theta^{max} - R_t & \text{If } R_t < \theta^{min} \\ 0 & \text{Otherwise.} \end{cases} \tag{1.10}$$

We might then set our sales quantity $x^s_t = \min\{R_t, D_t\}$, which means we satisfy as much demand as we can. This policy is parameterized by θ^{Ord}, so now we just have to find the values that produce the highest profit. This means we translate our objective function (1.9) to

$$\max_{\theta^{Ord}} \mathbb{E} \sum_{t=0}^{T} \big(p_t X^{\pi,s}_t(R_t|\theta^{Ord}) - c_t X^{\pi,o}_t(R_t|\theta^{Ord})\big). \tag{1.11}$$

Solving (1.11) may not be easy, but it is certainly easier to understand. For example, we could simply do a grid search over all possible values of θ (discretized), using our sampled approximation in (1.9). In fact, solving (1.11) is itself a stochastic optimization problem. Later we are going to describe a variety of ways of solving this problem, but an immediate approximation is to solve a sampled model as we did in equation (1.9), which would be stated as

$$\max_{\theta^{Ord}} \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T} \big(p_t X^{\pi,s}_t(R_t(\omega^n)|\theta^{Ord}) - c_t X^{\pi,o}_t(R_t(\omega^n)|\theta^{Ord})\big). \tag{1.12}$$

This “order up to” policy is a particularly simple example. It is known to work well under certain situations (in fact, it is widely used in practice). But inventory problems arise in many settings, and an order-up-to policy would not work well in settings with highly variable (yet predictable) demands. In such a setting, it makes more sense to use a lookahead policy, where we optimize a deterministic approximation over some appropriately chosen horizon H. For this, let

$\tilde{D}_{tt'}$ = The forecast of the demand $D_{t'}$ made at time t.

Now, we formulate a deterministic approximation of the problem at time t using

$$X_t^{LA}(S_t) = \underset{\tilde{x}_{tt'}, t'=t,\ldots,t+H}{\arg\max} \sum_{t'=t}^{t+H} \big(p_{t'}\tilde{x}^s_{tt'} - c_{t'}\tilde{x}^o_{tt'}\big). \tag{1.13}$$

Note that we use tilde's on all the variables in the lookahead model, to avoid confusion with the base model. These tilde-variables are also indexed by two time variables: the time t at which we are making the decision, and t' which is the point in time in the future within the lookahead model.

This problem is solved subject to modified versions of (1.2)-(1.6). For example, instead of using the actual demand D_t, we use the forecasted demand $f^D_{tt'}$ which is our best estimate of $D_{t'}$ given what we know at time t. The lookahead version of these constraints would be written

$$\begin{aligned}
\tilde{x}^s_{tt'} &\leq f^D_{tt'}, && (1.14)\\
\tilde{x}^s_{tt'} &\leq \tilde{R}_{tt'}, && (1.15)\\
\tilde{x}^s_{tt'} &\geq 0, && (1.16)\\
\tilde{x}^o_{tt'} &\geq 0, && (1.17)\\
\tilde{R}_{t,t'+1} &= \tilde{R}_{tt'} + \tilde{x}^o_{tt'} - \tilde{x}^s_{tt'}. && (1.18)
\end{aligned}$$

We are not quite done. In the lookahead model, all activities for $t' > t$ are basically forecasts. For example, we are not going to actually order $\tilde{x}^s_{tt'}$ for $t' > t$; we are simply creating a plan to help us make the best decision $\tilde{x}^s_{tt}$ which is what we are actually going to implement.

We now have two policies, $X^{Ord}(S_t|\theta^{Ord})$ and $X_t^{LA}(S_t)$, where $X^{Ord}(S_t|\theta^{Ord})$ is parameterized by θ^{Ord}. How do we decide which is best? We have to compute

$$\begin{aligned}
F^{Ord} &= \max_{\theta^{Ord}} \mathbb{E} \sum_{t=0}^{T} \big(p_t X_t^{Ord,s}(R_t|\theta^{Ord}) - c_t X_t^{Ord,o}(R_t|\theta^{Ord})\big), && (1.19)\\
F^{LA} &= \mathbb{E} \sum_{t=0}^{T} \big(p_t X_t^{LA,s}(R_t) - c_t X_t^{LA,o}(R_t)\big). && (1.20)
\end{aligned}$$

In other words, we compare the best policy within the class of order-up-to policies (by searching for the best value of θ^{Ord}), and then compare it to how well the lookahead policy works.

1.4.4 A generic formulation

Modeling sequential decision problems can be very subtle, especially considering the richness of the problem class. This book places considerable attention on careful modeling, but for the moment, we are going to briefly introduce some core elements. First, assume that $C(S_t, x_t)$ is the contribution (or cost) of being in a state S_t (this could be an inventory, or sitting at a node in the network, or holding an asset when the market price is at a certain level) and taking an action x_t (ordering more inventory, moving from one node to the next, or holding/selling the asset).

Next, we have to describe how our system evolves over time. If we are in some state S_t and make a decision x_t, next assume that we observe some new information W_{t+1} (this could be the demand for our product, the travel time over a congested link in the network, or the change in the asset price). The new information W_{t+1} (we call it the *exogenous information*) is random at time t but becomes known by time $t+1$. With this information, assume we have a transition function known as the *system model* $S^M(S_t, x_t, W_{t+1})$ that specifies the next state S_{t+1}, which we write using

$$S_{t+1} = S^M(S_t, x_t, W_{t+1}). \tag{1.21}$$

If we could fix the sequence $x = (x_0, x_1, \ldots, x_T)$ over the horizon from 0 to time period T, we could evaluate our solution using

$$\hat{F}(x) = \sum_{t=0}^{T} C(S_t, x_t), \tag{1.22}$$

where S_t evolves according to (1.21). Keeping in mind that the sequence $W_1, W_2, \ldots, W_T$ is a series of random variables (such as the orders for our inventory system), the total contribution $\hat{F}(x)$ in (1.22) is a random variable. We could take the expectation and then optimize over the vector x, giving us

$$F^* = \max_{x_0,\ldots,x_T} \mathbb{E} \sum_{t=0}^{T} C(S_t, x_t). \tag{1.23}$$

This would be like deciding in advance how much to order before you have learned any of the demands. This ignores our ability to adapt to the information as it arrives, captured by the state S_t of our system.

A more natural approach is to use one of our policies (such as our order-up-to policy $X_t^{Ord}(S_t)$, or our lookahead policy $X^{LA}(S_t)$). Let $X_t^\pi(S_t)$ be our generic policy. If we run a simulation of the policy (sampling or observing a single sequence $W_1, \ldots, W_T$), the performance of the policy would be given by

$$\hat{F}^\pi = \sum_{t=0}^{T} C(S_t, X_t^\pi(S_t)). \tag{1.24}$$

The quantity $\hat{F}^\pi$ is a random variable, since we would get different values each time we ran our simulation. Imagine we run our simulation $n = 1, \ldots, N$ times, and let $W_1^n, \ldots, W_T^n$ be the nth sequence of observations of the random information. Imagine this produces the sequence of states, actions and information

$$(S_0, x_0 = X^\pi(S_0), W_1^n, S_1^n, x_1^n = X^\pi(S_1^n), W_2^n, \ldots, S_t^n, x_t^n = X^\pi(S_t^n), W_{t+1}^n, \ldots)$$

We could then compute an average value of our policy using

$$\overline{F}^\pi = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T} C(S_t^n, X_t^\pi(S_t^n)). \tag{1.25}$$

While we often do this in practice, $\overline{F}^{\pi}$ is still a random variable (it depends on our sample of N sequences of random variables). We state the core problem using an expectation, which we write as

$$F^{\pi} = \mathbb{E}\left\{\sum_{t=0}^{T} C(S_t, X_t^{\pi}(S_t))|S_0\right\}. \tag{1.26}$$

This is how we go about evaluating a policy. Now we have to address the problem of designing policies.

1.5 SOLUTION STRATEGIES

At the heart of any stochastic optimization problem is an uncertainty operator such as an expectation (or risk measure) that introduces computational complexities. Our solution strategies are largely organized around how we handle the uncertainty operator, which can be organized along three broad lines:

Exact solution of original problem - Here, we exploit special structure to derive analytical solutions, or use numerical algorithms to exploit our ability to solve expectations exactly. In chapter 4, we describe a type of stochastic shortest path problem that reduces to a deterministic shortest path problem.

Exact solution of sampled problem - We can reduce uncertainty models that are computationally intractable (for example, a multivariate normal distribution) with a sampled approximation. Thus, if a random vector W has a mean vector $\mu_i = \mathbb{E}W_i$ and a covariance matrix Σ where $\Sigma_{ij} = Cov(W_i, W_j)$, then expectations involving W become intractable in more than two or three dimensions. However, we can replace this distribution with a sample $W^1, \ldots, W^K$, where we might then replace the original distribution with the discrete sample where we might assume that $Prob[W = W^k] = 1/K$ (for example).

Adaptive learning algorithms - Some of the most powerful strategies use iterative, sampling-based algorithms that work with the original probability model, applying iterative algorithms that work with one sample at a time. We may try to design algorithms that might give the optimal solution asymptotically, or a "good" solution within a fixed computational budget.

We cover all three strategies in this volume. Exact solutions are limited to very special cases, but these are sometimes very important special cases that provide a foundation to more complex problems. Sampled solution strategies represent a powerful and popular strategy, but can lead to the need to solve very large problems. However, most of our attention will focus on iterative algorithms which lend themselves to the richest problem classes.

1.6 DESIGNING POLICIES FOR SEQUENTIAL DECISION PROBLEMS

What often separates one field of stochastic programming (such as "stochastic programming") from another (e.g. ”dynamic programming") is the type of policy that is used to solve a problem. Possibly the most important aspect of our unified framework in this volume is how we have identified and organized different classes of policies.

The entire literature on stochastic optimization can be organized along two broad strategies for creating policies:

Policy search - This includes all policies where we need to search over a set of parameters θ to maximize (or minimize) an objective function such as (1.11) or the sampled version (1.12).

Policies based on lookahead approximations - Here we are going to build our policy so that we make the best decision now given an estimate (which is typically approximate) of the downstream impact of the decision.

Our order-up-to policy $X^{Ord}(S_t|\theta^{Ord})$ is a nice example of a policy that has to be optimized (we might say tuned). The optimization can be done using a simulator, as is implied in equation (1.12). The lookahead policy $X_t^{LA}(S_t)$ in (1.13) uses an approximation of the future to find a good decision now.

1.6.1 Policy search

Policies determined through policy search can be divided into two classes:

Policy function approximations (PFAs) - These are analytical functions that map a state (which includes all the information available to us) to an action.

Cost function approximations (CFAs) - These policies minimize or maximize some analytical function (this might include analytically modified constraints).

A good example of a PFA is our order-up-to policy as a *policy function approximation*. PFAs come in many forms, but these can be divided into three major classes:

Lookup tables - These are used when a discrete state S can be mapped to a discrete action, such as:

- When you order red meat for dinner, you drink red wine; if you order chicken or fish, you drink white wine.
- If the amount of cash in a mutual fund is R_t, the stock market index is M_t and interest rates are I_t, then invest x_t into equities ($x_t < 0$ means withdraw from equities).

Parametric functions - These describe any analytical functions parameterized by a vector of parameters θ. Our order-up-to policy is a simple example. We might also write it as a linear model. Let S_t be a state variable that captures all the information we have (and need) to make a decision, and let $(\phi_f(S_t))_{f\in\mathcal{F}}$ be a set of features. For example, our cash in a mutual fund problem has $S_t = (R_t, M_t, I_t)$, we could create features $\phi_1(S_t) = R_t, \phi_2(S_t) = R_t^2, \phi_3(S_t) = M_t, \phi_4(S_t) = \log(I_t)$. We might then write our policy as

$$X^{PFA}(S_t|\theta) = \theta_1\phi_1(S_t) + \theta_2\phi_2(S_t) + \theta_3\phi_3(S_t) + \theta_4\phi_4(S_t).$$

Other examples of parametric functions include nonlinear models such as neural networks and logistic regression.

Nonparametric functions - These include functions that might be locally smoothed estimates (known as kernel regression or k-nearest neighbor), or locally parametric models, which might be linear models for different (possibly pre-specified) regions. More advanced nonparametric models include support vector machines (we describe these in much more detail in chapter 3).

The second class of functions that can be optimized using policy search is called parametric *cost function approximations*, or CFAs. CFAs are distinguished by the need to minimize or maximize some function, typically subject to some set of (possibly complex) constraints. For example, we might specify a feasible region $\mathcal{X}_t$ by a system of linear equations

$$\mathcal{X}_t = \{x | A_t x_t = b_t, x_t \geq 0, x_t \leq u_t\}.$$

Now imagine that we start with a simple myopic policy that tries to maximize contributions now, ignoring the impact on the future. We would write this policy as

$$X^{myopic}(S_t) = \arg\max_{x_t \in \mathcal{X}_t} c_t x_t. \tag{1.27}$$

We can modify this policy using a parametric term in the objective, which we might write as

$$X^{myopic-CFA}(S_t|\theta) = \arg\max_{x_t \in \mathcal{X}_t} \left(c_t x_t + \sum_{f \in \mathcal{F}} \theta_f \phi_f(S_t, x_t) \right), \tag{1.28}$$

where $(\phi_f(S_t, x_t))_{f \in \mathcal{F}}$ is a set of features drawn from the state S_t and the decision vector x_t.

Both PFAs and CFAs have to be tuned using equation (1.29). Imagine that policy π has a tunable parameter vector θ, so we write the dependence of the policy on θ using $X^\pi_t(S_t|\theta)$. We now write the problem of tuning θ using

$$F^\pi = \max_\theta \mathbb{E} \left\{ \sum_{t=0}^{T} C(S_t, X^\pi_t(S_t|\theta)) | S_0 \right\}. \tag{1.29}$$

Thus, F^π is the best performance of the policy π when its parameters are tuned to maximize performance.

1.6.2 Policies based on lookahead approximations

A natural strategy for making decisions is to consider the downstream impact of a decision you make now. There are two ways of doing this:

Value function approximations (VFAs) - Assume that you are in a state S_t (a node in a network, the level of inventory), take a decision x_t (or action a_t) (which takes you to another node, or changes the inventory level), which produces a contribution (or cost) $C(S_t, x_t)$ and you then transition to a state S_{t+1} (the downstream node, or the new inventory level). If you have an estimate of the value $V_{t+1}(S_{t+1})$ of starting in state S_{t+1} and then proceeding until the end of the horizon, we can use this value function to make the best decision. If the downstream state S_{t+1} is a deterministic function of the starting state S_t and the action a_t, then we can define a policy based on value functions using

$$X^{VFA}(S_t) = \arg\max_{x_t} \big(C(S_t, x_t) + V_{t+1}(S_{t+1})\big).$$

We are primarily interested in problems where the transition to the downstream state S_{t+1} involves a random variable (such as the demand in an inventory problem), which means we have to use an expectation

$$X^{VFA}(S_t) = \arg\max_{x_t} \big(C(S_t, x_t) + \mathbb{E}\{V_{t+1}(S_{t+1})|S_t\}\big).$$

While there are special cases where we can compute these value functions exactly, most of the time they are approximations called *value function approximations* (or VFAs), producing what we call VFA-based policies.

Direct lookahead models (DLAs) - The second approach avoids the need to create an analytical function $V_t(S_t)$, and instead tries to optimize over the entire horizon from time t until the end of the planning horizon T (or over some horizon $(t, t+H)$. While this is often (but not always) possible with deterministic models, there are only a few special cases where it can be done for stochastic models, requiring that we design and solve approximate lookahead models. The deterministic lookahead model $X_t^{LA}(S_t)$ introduced above is a deterministic approximation of a stochastic model.

Unlike PFAs and CFAs, policies based on direct lookahead do not have to be tuned using optimization models such as that depicted in equation (1.29). In a very small number of cases, we can compute the value function exactly, or solve the lookahead model exactly (without introducing approximations), which means we can find optimal policies. We note that there are also special cases where the optimal policy has a known, parametric form, which means that solving equation (1.29) for the best PFA or CFA can also provide an optimal policy.

1.6.3 Mixing and matching

It is possible to create hybrids. For example, we may wish to modify our lookahead policy to recognize that $f^D_{tt'}$ is just a forecast of $D_{t'}$, which means we have to think about the impact of the uncertainty in the forecast. For example, we might want to factor our forecasts as a way of hedging uncertainty. Let θ^{LA}_τ be our factor for the forecast τ periods into the future, where we are going to replace $f^D_{tt'}$ with $\theta^{LA}_{t'-t} f^D_{tt'}$. If $\theta^{LA}_{t'-t} < 1$ then it means we are using a more conservative forecast. Thus, we might replace (1.14) with

$$x^s_{tt'} \quad \leq \quad \theta^{LA}_{t'-t} f^D_{tt'}. \tag{1.30}$$

Now we have to determine the factors $\theta^{LA} = (\theta^{LA}_1, \ldots \theta^{LA}_H)$.

Unlike the cost function approximation introduced in equation (1.28), where we introduced a parametric modification of the objective function, here we have parametrically modified the constraints (as in equation (1.30)). Although this is a lookahead model, it is also a cost function approximation, which means the parameters have to be tuned using policy search by solving

$$F^{LA-CFA} \quad = \quad \max_{\theta^{LA}} \mathbb{E} \sum_{t=0}^{T} \big(p_t X_t^{LA,s}(R_t|\theta^{LA}) - c_t X_t^{LA,o}(R_t|\theta^{LA})\big). \tag{1.31}$$

We did not say this problem is easy, but it is well defined, and there are different algorithmic strategies that we can draw on to help us with this (we discuss these in more depth in chapters 5 and 7).

1.6.4 Pulling it all together

In practice, problems where it is possible to find optimal policies are rare. For this reason, solving sequential decision problems almost invariably involves a search over classes of suboptimal policies. Historically, this has been done by pursuing the problem from the perspective of the different communities. However, all of the approaches used in these different communities boil down to the four classes of policies we have introduced above: policy function approximations (PFAs), cost function approximations (CFAs), policies based on value function approximations (VFAs), and direct lookahead policies (DLAs).

Complicating the problem of organizing the material in a natural progression was the realization that virtually every problem can be formulated as a sequential decision problem (that is, a dynamic program), even when the basic problem (such as the simple newsvendor problem in (1.1)) is static. We can solve pure learning problems (no physical state) or problems with pure physical states (e.g. blood management) as sequential problems, which can be reasonably solved using all four classes of policies.

Despite this merger of these disparate problems, pure learning problems are the simplest to present. These are problems where the only state variable is the belief state. We use this problem class to establish some basic modeling and algorithmic strategies. We then transition to problems with physical states, since this opens the door to much more complex problems. Finally, we transition to the much harder problems which combine physical states with knowledge states. In between are many variations that involve exogenous information states, and high dimensional problems that exploit convexity.

1.7 A SEQUENCE OF PROBLEM CLASSES

Eventually, we are going to show that most stochastic optimization problems can be formulated using a common framework. However, this seems to suggest that all stochastic optimization problems are the same, which is hardly the case. It helps to identify major problem classes.

- Deterministically solvable problems - These are optimization problems where the uncertainty has enough structure that we can solve the problem exactly using deterministic methods. This covers an important class of problems, but we are going to group these together for now. All remaining problem classes require some form of adaptive learning.

- Pure learning problems - We make a decision x^n (or x_t), then observe new information W^{n+1} (or W_{t+1}), after which we update our knowledge to make a new decision. In pure learning problems, the only information passed from iteration n to $n+1$ (or from time t to time $t+1$) is updated knowledge, while in other problems, there may be a physical state (such as inventory) linking decisions.

- Stochastic problems with a physical state - Here we are managing resources, which arise in a vast range of problems where the resource might be people, equipment, or inventory of different products. Resources might also be money or different types of financial assets. There are a wide range of physical state problems depending on the nature of the setting. Some major problem classes include

Stopping problems - The state is 1 (process continues) or 0 (process has been stopped). This arises in asset selling, where 1 means we are still holding the asset, and 0 means it has been sold.

Inventory problems - We hold a quantity of resource to meet demands, where leftover inventory is held to the next period.

Inventory problems with dynamic attributes - A dynamic attribute might be spatial location, age or deterioration.

Inventory problems with static attributes - A static attribute might reflect the type of equipment or resource which does not change.

Multiattribute resource allocation - Resources might have static and dynamic attributes, and may be re-used over time (such as people or equipment).

Discrete resource allocation - This includes dynamic transportation problems, vehicle routing problems, and dynamic assignment problems.

- Physical state problems with an exogenous information state - While managing resources, we may also have access to exogenous information such as prices, weather, past history, or information about the climate or economy. Information states come in three flavors:
  - Memoryless - The information I_t at time t does not depend on past history, and is "forgotten" after a decision is made.
  - First-order exogenous process - I_t depends on I_{t-1}, but not on previous decisions.
  - State-dependent exogenous process - I_t depends on S_{t-1} and possibly x_{t-1}.
- Physical state with a belief state - Here, we are both managing resources while learning at the same time.

This list provides a sequence of problems of increasing complexity.

1.8 BRIDGING TO STATISTICS

Finding the best policy is the same as finding the best function that achieves the lowest cost, highest profits or best performance. Stochastic optimization is not the only community that is trying to find the best function. Another important community is statistics and machine learning, where a common problem is to use a dataset (y^n, x^n), where $x^n = (x_1^n, \ldots, x_K^n)$ to predict y^n. For example, we might specify a linear function of the form:

$$y^n = f(x^n|\theta) = \theta_0 + \theta_1 x_1^n + \ldots + \theta_K^n x_K + \epsilon_n, \tag{1.32}$$

where ϵ_n is a random error term that is often assumed to be normally distributed with mean 0 and some variance σ^2.

We can find the parameter vector $\theta = (\theta_1, \ldots, \theta_K)$ by solving

$$\min_\theta \frac{1}{N} \sum_{n=1}^{N} \left(y^n - f(x^n|\theta)\right)^2. \tag{1.33}$$

	Statistical learning	Stochastic optimization
(1)	Batch estimation: $\min_\theta \frac{1}{N}\sum_{n=1}^N (y^n - f(x^n\|\theta))^2$	Sample average approximation: $x^* = \arg\max_{x\in\mathcal{X}} \frac{1}{N} F(x, W(\omega^n))$
(2)	Online learning: $\min_\theta \mathbb{E}F(Y - f(X\|\theta))^2$	Stochastic search: $\min_\theta \mathbb{E}F(X, W)$
(3)	Searching over functions: $\min_{f\in\mathcal{F},\theta\in\Theta^f} \mathbb{E}F(Y - f(X\|\theta))^2$	Policy search: $\min_\pi \mathbb{E}\sum_{t=0}^T C(S_t, X^\pi(S_t))$

Table 1.1 Comparison of classical problems faced in statistics (left) versus similar problems in stochastic optimization (right).

Our problem of fitting a model to the data, then, involves two steps. The first is to choose the function $f(x|\theta)$, which we have done by specifying the linear model in equation (1.32) (note that this model is called "linear" because it is linear in θ). The second step involves solving the optimization problem given in (1.33), which precisely mirrors the optimization problem in (1.12). The only difference is the specific choice of performance metric.

It is common, in both stochastic optimization and statistics, to pick a function such as (1.32) or our order-up-to policy in (1.10), and then use an equation such as (1.33) or (1.12) to find the best parameters. Of course, both communities would like to have a method that would search over classes of functions, as well as the parameters that define a particular function class. The stochastic optimization community has proceeded by tailoring specific function classes that seem to be well suited to specific classes of problems. The statistics/machine learning community has more of a culture of trying different techniques on generic datasets, although there is still an appreciation of the characteristics of the data guiding the choice of model.

Table 1.1 provides a brief comparison of some major problem classes in statistical learning, and corresponding problems in stochastic optimization. As of this writing, we feel that the research community has only begun to exploit these links, so we ask the reader to be on the lookout for opportunities to help build this bridge.

1.9 FROM DETERMINISTIC TO STOCHASTIC OPTIMIZATION

There is a long, rich and tremendously successful tradition in the development of deterministic optimization models and algorithms. The first and most visible success was in the formulation (in tne 1930's) and solution (in the 1940's and 1950's) of linear programs of the general form

$$\min_x \sum_{i=1}^n c_i x_i,$$

subject to

$$\begin{aligned} Ax &= b, \\ x &\geq 0. \end{aligned}$$

By the late 1980's to early 1990's, commercial packages such as Cplex (and later Gurobi), combined with the advancing power of computers, had evolved into an exceptionally powerful tool that could reliably solve a very broad class of problems. Close behind were the dramatic successes solving problems where some or all of the elements of the vector x had to be integer (initially 0 or 1, but then handling general integers). From the 1980's when integer programs were limited to perhaps a dozen integer variables, by 2000 we could solve many classes of integer programs with tens of thousands of integer variables (problem structure affects this tremendously). Similar advances have been made with different classes of nonlinear programs.

All optimization problems involve a mixture of modeling and algorithms. With integer programming, modeling is important (especially for integer problems), but modeling has always taken a back seat to the design of algorithms. A testament of the power of modern algorithms is that they generally work well (for a problem class) with modest expertise in modeling strategy.

Stochastic optimization is different.

Figure 1.3 illustrates some of the major differences between how we approach deterministic and stochastic optimization problems.

Models - Deterministic models are systems of equations. Stochastic models are often complex systems of equations, numerical simulators, or even physical systems with unknown dynamics.

Objective - Deterministic models minimize or maximize some well defined metric such as cost or profit. Stochastic models require that we deal with statistical performance measures and uncertainty operators such as risk

Searching for - In deterministic optimization, we are looking for a deterministic scalar or vector. In stochastic optimization, we are almost always looking for functions that we will refer to as policies.

Goal - The goal of deterministic optimization is to find an optimal decision. Most of the time the challenge in stochastic optimization is to find an optimal policy, which is a function.

What is hard - The challenge of deterministic optimization is designing an effective algorithm. The hardest part of stochastic optimization, by contrast, is the modeling. Designing and calibrating a stochastic model can be surprisingly difficult. Optimal policies are rare, and a policy is not optimal if the model is not correct.

1.10 PEDAGOGY

The book is organized into six parts, as follows:

Part I - Introduction and foundations - We start by providing a summary of some of the most familiar canonical problems, followed by an introduction to approximation strategies which we draw on throughout the book.

- Canonical problems (chapter 2) - We begin by listing a series of canonical problems that are familiar to different communities, primarily using the notation familiar to those communities.
- Learning in stochastic optimization (chapter 3) - So many approaches in stochastic optimization involve some form of statistical approximation using machine

Figure 1.3 Deterministic vs. stochastic optimization

	Deterministic	Stochastic
Models	System of equations	Complex functions, numerical simulations, physical systems
Objective	Minimize cost	Policy evaluation, risk measures
Searching for	Real-valued vectors	Functions (policies)
Goal	Finding optimal decision	Finding optimal policies
What is hard	Designing algorithms	Modeling

learning that we felt it best to start by giving an overview of some of the major statistical learning approaches, where we focus primarily (if not exclusively) on recursive methods.

Part II - Learning problems - These are stochastic optimization problems that can be solved using an adaptive algorithm where the only information linking iterations is the belief about the function. We also call these state-independent functions, to distinguish them from the more general state-dependent functions we handle starting in Part III.

- Introduction to stochastic optimization (chapter 4) - We begin with a problem we call the *basic stochastic optimization problem* which provides the foundation for most stochastic optimization problems. In this chapter we also provide examples of how some problems can be solved exactly. We then introduce the idea of solving sampled models before transitioning to adaptive learning methods.
- Derivative-based stochastic optimization (chapter 5) - Derivative-based algorithms represent one of the earliest adaptive methods proposed for stochastic optimization. These methods form the foundation of what is classically referred to as stochastic search.
- Stepsize policies (chapter 6) - Throughout the use of sampling-based algorithms is the need to perform smoothing between old and new estimates using what are commonly known as stepsizes (some communities refer to these as learning rates). Stepsize policies (which are more typically known as stepsize rules) are functions for determining stepsizes based on the state of the dynamic system, which is our stochastic learning algorithm.
- Derivative-free stochastic optimization (chapter 7) - We then transition to derivative-free stochastic optimization, which encompasses a variety of fields with names such as ranking and selection (for offline learning) and multiarmed bandit problems (for online, or cumulative reward, formulations).

Part III - State-dependent functions - Here we transition to the much richer class of sequential problems where the function being optimized is state dependent. These problems may or may not have a belief state.

- State-dependent applications (chapter 8) - We begin our presentation with a series of applications of problems where the function is state dependent. State variables can arise in the objective function (e.g. prices), but in most of the applications the state arises in the constraints, which is typical of problems that involve the management of physical resources.
- Modeling general dynamic programs (chapter 9) - This chapter provides a comprehensive summary of how to model general (state-dependent) sequential decision problems in all of their glory.
- Modeling uncertainty (chapter 10) - To find good policies (to make good decisions), you need a good model, and this means an accurate model of uncertainty. In this chapter we identify different sources of uncertainty and discuss how to model them.
- Policies (chapter 11) - Here we provide a more comprehensive overview of the different strategies for creating policies, leading to the four classes of policies that we first introduce in part I for learning problems.

Part IV - Policy search - These chapters describe policies that have to be tuned, either in a simulator or through experience.

- Policy function approximations (chapter 12) - In this chapter we consider the use of parametric functions (plus some variations) for directly approximating policies. We search over a well-defined parameter space to find the policy that produces the best performance. PFAs are well suited to problems with scalar action spaces, or low-dimensional continuous actions.
- Parametric cost function approximations (chapter 13) - This strategy is suited for high-dimensional stochastic optimization problems that require the use of solvers for linear, integer or nonlinear programs. This policy class has been overlooked in the research literature, but is widely used (heuristically) in industry.

Part V - Policies based on lookahead approximations - Policies based on lookahead approximations are the counterpart to policies derived from policy search. Here, we design good policies by understanding the impact of a decision now on the future. We can do this by finding (usually approximately) the value of being in a state, or by planning over some horizon.

- Discrete Markov decision processes (chapter 14) - There is a rich and elegant theory for dynamic programs that are described by discrete states and actions, where the number of states and actions is not too large. This is one of the very important special cases where the problem can be solved optimally.
- Structured dynamic programs (chapter 15) - There are other dynamic programs that can be solved optimally by exploiting special structure. A major example is optimal control problems with quadratic cost functions, but there are others.
- Backward approximate dynamic programming (chapter 16) - Classical backward dynamic programming methods (which we used in the previous two chapters) can quickly become computationally intractable, especially when using discrete states and actions, which are sensitive to the number of dimensions in the state variable (primarily). Before we completely give up on backward

dynamic programming, we describe methods where we can do this approximately.

- Foward approximate dynamic programming I: The value of a policy (chapter 17) - This is the first step using machine learning methods to approximate the value of policy as a function of the starting state, which is the foundation of a broad class of methods known as approximate (or adaptive) dynamic programming, or reinforcement learning.
- Foward approximate dynamic programming II: Policy optimization (chapter 18) - In this chapter we build on foundational algorithms such as Q-learning, value iteration and policy iteration, first introduced in chapter 14, to try to find high quality policies based on value function approximations.
- Forward approximate dynamic programming III: Convex functions (chapter 19) - This chapter focuses on convex problems, with special emphasis on stochastic linear programs. Here we exploit convexity to build high quality approximations, where we emphasize the use of a powerful technique known as Benders decompositions.
- Lookahead policies (chapter 20) - Often called *model predictive control*, lookahead policies are what you do when all else fails (the problem is that all else fails fairly frequently). We have already seen one-step lookahead policies in chapter 7, so here we focus on multi-step lookahead policies, where the discussion is divided between problems with discrete actions (popular in computer science), and those with vector-valued controls. We also consider both deterministic lookahead (often called model predictive control or rolling/receding horizon procedures) and more complex policies that use stochastic lookahead models.

Part VI - Risk - Risk is the next frontier in stochastic optimization.

- Risk and robust optimization (chapter 21) - An emerging area of research involves the recognition that when uncertainty is involved, we often are more interested in extreme events than averages. This is captured through the use of risk measures, or, in some settings, the worst case, an area that has become known as *robust optimization*.

1.11 BIBLIOGRAPHIC NOTES

- Section xx -

PROBLEMS

1.1 What are the three classes of state variables? Explain the differences.

1.2 What is meant by a “model-free” dynamic program?

1.3 What are the two strategies for designing policies for sequential decision problems? Give the basic equation for each.

1.4 Consider an asset selling problem where you need to decide when to sell an asset. Let p_t be the price of the asset if it is sold at time t, and assume that you model the evolution of the price of the asset using

$$p_{t+1} = p_t + \theta(p_t - 60) + \varepsilon_{t+1},$$

We assume that the noise terms $\varepsilon_t,\ t = 1, 2, \ldots$ are independent and identically distributed over time, where $\varepsilon_t \sim N(0, \sigma^2_\varepsilon)$. Let

$$R_t = \begin{cases} 1 & \text{If we are still holding the asset at time } t, \\ 0 & \text{Otherwise.} \end{cases}$$

Further let

$$x_t = \begin{cases} 1 & \text{If we sell the asset at time } t, \\ 0 & \text{Otherwise.} \end{cases}$$

Of course, we can only sell the asset if we are still holding it. We now need a rule for deciding if we should sell the asset. We propose

$$X^\pi(S_t|\rho) = \begin{cases} 1 & \text{If } p_t \geq \bar{p}_t + \rho \text{ and } R_t = 1, \\ 0 & \text{Otherwise.} \end{cases}$$

where

$$\begin{aligned} S_t &= \text{The information we have available to make a decision (we have to, design this),} \\ \bar{p}_t &= .9\bar{p}_{t-1} + .1p_t. \end{aligned}$$

a) What are the elements of the state variable S_t for this problem?

b) What is the uncertainty?

c) Imagine running a simulation in a spreadsheet where you are given a sample realization of the noise terms over T time periods as $(\hat{\varepsilon})^T_{t=1} = (\hat{\varepsilon}_1, \hat{\varepsilon}_2, \ldots, \hat{\varepsilon}_T)$. Note that we treat $\hat{\varepsilon}_t$ as a number, such as $\hat{\varepsilon}_t = 1.67$ as opposed to ε_t which is a normally distributed random variable. Write an expression for computing the value of the policy $X^\pi(S_t|\rho)$ given the sequence $(\hat{\varepsilon})^T_{t=1}$. Given this sequence, we could evaluate different values of ρ, say $\rho = 0.75, 2.35$ or 3.15 to see which performs the best.

d) In reality, we are not going to be given the sequence $(\hat{\varepsilon})^T_{t=1}$. Assume that $T = 20$ time periods, and that

$$\begin{aligned} \sigma^2_\varepsilon &= 4^2, \\ p_0 &= \$65, \\ \theta &= 0.1. \end{aligned}$$

Write out the value of the policy as an expectation (see section 1.3).

e) Develop a spreadsheet to create 10 sample paths of the sequence $(\varepsilon_t),\ t = 1, \ldots, 20)$ using the parameters above. You can generate a random observation of ε_t using the function `NORM.INV(RAND(),0,`σ`)`. Let the performance of our decision rule $X^\pi(S_t|\rho)$ be given by the price that it decides to sell (if it decides to sell), averaged over all 10 sample paths. Now test $\rho = 1, 2, 3, 4, ..., 10$ and find the value of ρ that seems to work the best.

f) Repeat (e), but now we are going to solve the problem

$$\max_{x_0,\ldots,x_T} \mathbb{E} \sum_{t=0}^{T} p_t x_t.$$

We do this by picking the time t when we are going to sell (that is, when $x_t = 1$) before seeing any information. Evaluate the solutions $x_2 = 1, x_4 = 1, \ldots, x_{20} = 1$. Which is best? How does its performance compare to the performance of $X^\pi(S_t|\rho)$ for the best value of ρ?

g) Finally, repeat (f), but now you get to see all the prices and then pick the best one. This is known as a *posterior bound* because it gets to see all the information in the future to make a decision now. How do the solutions in parts (e) and (f) compare to the posterior bound? (There is an entire field of stochastic optimization that uses this strategy as an approximation.)

h) Classify the policies in (e), (f) and (g) (yes, (g) is a class of policy) according to the classification described in section 1.5 of the text.

CHAPTER 2

CANONICAL PROBLEMS AND APPLICATIONS

It helps when learning a field such as optimization under uncertainty to have example problems to relate to. This chapter lists a wide range of problems with which we have worked on. We encourage readers to at least skim these, and to pay attention to problem characteristics. One important problem feature is the dimensionality of the decisions, which are either discrete (sometimes binomial) action spaces, or possibly high-dimensional vectors.

Another important but more subtle dimension is the nature of the state variable, which may consist of some combination of physical state variables, informational state variables, and distributional information (belief state). Different classes of state variables include the following combinations:

Pure learning - These are problems where we are just trying to learn an unknown function $f(x)$. Examples include:

- The function $f(x) = \mathbb{E}F(x, W)$ where $F(x, W)$ is known, but either the distribution of W is unknown, or the expectation is not computable (for example, W might be multi-dimensional).
- The function $f(x)$ is unknown. Examples include the behavior of a complex, black box simulation model (for example, of the operations of a company, or the dynamics of a chemical process), or an actual physical system (for example, estimating the market response to a price, or measuring the reduction in CO2 resulting from restrictions on generation from coal plants).

What characterizes these problems is that we can choose x^n, learn a noisy observation $\hat{f}^n$, and then try to update our belief $\overline{F}^n(x)$ about $f(x)$. All we carry from iteration n to $n+1$ is our updated belief $\overline{F}^{n+1}$.

Pure physical state - There are many problems where we are managing resources. It might be a single resource such as a vehicle moving over a graph, or an inventory system, or even a robot. Typically the decisions we can make at time t depend on the state of the resource.

Physical plus information state - We may be managing a set of resources (water in a reservoir, a fleet of driverless taxis, investments in stock funds) which evolves in a way that reflects other sources of information such as weather, humidity, or information about the economy. There are several important classes of information process:

- Independent of past history - The customers available to a taxi just after she finishes a trip may be completely independent of past history. These processes may be called "zeroth order Markov," "memoryless," i.i.d. (independent and identically distributed, although the key description is independent), or they may be described as exhibiting "intertemporal independence."
- First order Markov - These are the most common information processes, and describe problems where the information I_{t+1} is conditionally dependent only on I_t.
- History-dependent - This generalizes the first-order Markov process, and expresses problems where the distribution of I_{t+1} depends on the entire history.

Physical/information plus learning - There are problems where our distribution of belief about functions or processes evolve as information is observed. These generally apply to problems where we are making observations from an exogenous source (field experiments, laboratory simulations) where the underlying distribution is unknown (this describes all the pure learning problems).

We encourage readers to try to classify each sample problem in terms of the problem types we first introduced in section 1.3, as well as the variations listed above.

2.1 CANONICAL PROBLEMS

Each community in stochastic optimization has a canonical problem that they use to illustrate their problem domain. Often, these canonical problems lend themselves to an elegant solution technique which then becomes a hammer looking for a nail. While these tools are typically limited to a specific problem class, they often illustrate important ideas that become the foundation of powerful approximation methods. For this reason, understanding these canonical problems helps to provide an important foundation for stochastic optimization.

2.1.1 Stochastic search

As we are going to learn, if there is a single problem that serves as a single umbrella for almost all stochastic optimization problems (at least, all the ones that use an expectation), it is a problem that is often referred to as stochastic search, which is written

$$\max_x \mathbb{E}F(x, W), \tag{2.1}$$

where x is a deterministic variable (or vector), and where the expectation is over the random variable W. Some authors like to make the random variable explicit by writing

$$\max_x \mathbb{E}_W F(x, W).$$

While probabilists frown on this habit, any notation that improves clarity should be encouraged. We are also going to introduce problems where it is useful to express the dependence on an initial state variable S_0, which is done by writing

$$\max_x \mathbb{E}\{F(x, W)|S_0\} \quad = \quad \mathbb{E}_{S_0}\mathbb{E}_{W|S_0}F(x, W). \tag{2.2}$$

Initial state variables can express the dependence of the problem on either deterministic or probabilistic information (say, a distribution about an unknown parameter). There are problems where the initial state S_0 may change each time we solve a problem. For example, S_0 might capture the medical history of a patient, after which we have to make a decide on a course of treatment, and then we observe medical outcomes. We will sometimes use the style in (2.1) for compactness, but as a general style, we are going to use (2.2) as our default style (the motivation for this becomes more apparent when you start working on real applications).

We assume that the expectation cannot be computed, either because it is computationally intractable, or because the distribution of W is unknown (but can be observed from an exogenous source). We are going to refer to (2.1) as the *asymptotic reward* version of our stochastic search problem.

This basic problem class comes in a number of flavors, depending on the following:

- Initial state S_0 - The initial state will include any deterministic parameters, as well as initial distributions of uncertain parameters. If S_0 is a set of fixed, deterministic parameters, we typically ignore it. However, there are problems where the initial state changes from one problem instance to another.

- Decision x - x can be binary, discrete (and finite, and not too large), categorical (finite, but a potentially very large number of choices), continuous (scalar or vector), or a discrete vector.

- Random information W - The distribution of W may be known or unknown, and the distribution can take on a variety of distributions ranging from nice distributions such as the normal or exponential, or one with heavy tails, spikes, and rare events.

- The function $F(x, W)$ may be characterized along several dimensions:
  - Derivative-based or derivative-free - If we can compute gradients $\nabla_x F(x, W)$ given W, then we have access to derivatives (known as stochastic gradients, since they depend on the information W) and can design algorithms accordingly. If we do not have access to derivatives, then we assume that we can observe samples of $F(x, W)$.
  - The cost of a function evaluation - The function $F(x, W)$ may be easy to evaluate (fractions of a second to seconds), or more expensive (minutes to hours to days to weeks).

- Search budget - May be finite (for example, we are limited to N evaluations of the function or its gradient), or infinite (obviously this is purely for analysis purposes - real budgets are always finite). There are even problems where a rule determines when we stop, which may be exogenous or dependent on what we have learned (these are called *anytime problems*).
- Final-reward vs. cumulative reward - Equation (2.1) is the final-reward version, which means that we only care about how well a solution x performs at the end of the search. Often, we are going to need to search for the best answer, and we may care about how well we do while we are searching, in which case we obtain the *cumulative reward* formulation.

Problem (2.1) is the asymptotic form of the basic stochastic optimization problem. In both chapters 5 and 7, we are going to introduce and analyze algorithms that asymptotically will return deterministic solutions x^* that solve (2.1) using derivative-based and derivative-free algorithmic strategies, respectively. Eventually we will show that we can also reduce sequential decision problems to this format, but this requires learning how to interpret x as a function that serves as a policy for making decisions.

2.1.1.1 Terminal reward formulation We start with what we are going to call the terminal reward formulation of our stochastic search problem. This is often referred to as the ranking and selection problem.

Let S^n represent what we know about the function after n iterations, and let $X^\pi(S^n)$ be the rule we use to choose the next point $x^n = X^\pi(S^n)$ to test, after which we observe W^{n+1} (we may just be able to observe $F^n = F(x^n, W^{n+1})$). We show later that our rule $X^\pi(S^n)$ may be an algorithm, or what we are going to characterize as a policy. Regardless of how we describe it, the policy is a function that uses what we know to determine what point to test x.

Imagine now that we fix our "policy" $X^\pi(S^n)$, and then use this to observe $W^1, W^2, \ldots,$ W^N, producing a final decision $x^{\pi,N}$ (we have no problem if the outcome W^n depends on the previous decision x^{n-1}). For example, we might be looking to find the best capacity for a battery or transmission line, the capacity of a dam, or even the strength of an airplane wing, all of which represent decisions that have to be made before any information becomes known. Since $x^{\pi,N}$ depends on our realizations of $W^1, W^2, \ldots, W^N$, it is a random variable.

This problem requires that we recognize possibly four sets of random variables:

- The initial state S_0, which may include distributions about uncertain parameters.
- The sequence of observations $W^1, \ldots, W^N$ guided by our policy $X^\pi(S^n)$.
- The implementation decision $x^{\pi,N}$ determined after N experiments.
- The uncertainty $\widehat{W}$ when evaluating the function given $x^{\pi,N}$.

Recognizing all these sources of uncertainty means that we can expand the expectation in (2.2) to obtain the objective function

$$\max_\pi \mathbb{E}_{S_0} E_{W^1,\ldots,W^N|S_0} E_{x^{\pi,N}|W^1,\ldots,W^N} E_{\widehat{W}|x^{\pi,N}} F(x^{\pi,N}, \widehat{W}). \tag{2.3}$$

We refer to (2.3) as the *terminal reward* version of our basic stochastic optimization problem since we are only evaluating the final implementation decision $x^{\pi,N}$.

A special case of our terminal reward formulation is when x is limited to a finite set of alternatives $\mathcal{X} = \{x_1, x_2, \ldots, x_M\}$, where M is the number of alternatives. The terminal reward problem defined over the finite set $\mathcal{X}$ is known as the *ranking and selection* problem, a problem class that has been studied since the 1950's.

The ranking and selection problem became quite popular in what is known as the "simulation-optimization" community, a group of people who use Monte Carlo simulation (which we introduce in some depth in chapter 10) to evaluate complex systems. A popular early application was to model the flow of jobs in a manufacturing operation that makes different component that have to move from one process to another. The problem would be to evaluate different designs. The earliest simulation optimization problems involved evaluating perhaps dozens of different design configurations. Early computers could run just one simulation at a time, and run times were much longer than they are now. The problem was to determine how much time to allocate to evaluating each design.

2.1.1.2 Cumulative reward Now assume that we want to find a policy that maximizes our total reward over our horizon of N days of sales, which we write as

$$\max_\pi \mathbb{E}_{W^1,\ldots,W^N} \sum_{n=0}^{N-1} F(X^\pi(S^n), W^{n+1}). \tag{2.4}$$

We refer to (2.4) as the *cumulative reward* version of our elementary stochastic optimization problem. If $x = X^\pi(S^n)$ belongs to a discrete set $\mathcal{X} = \{x_1, \ldots, x_M\}$, then this would be called the *multiarmed bandit problem.*

We show how we can close the circle if we let

$$G(x, Z) = \sum_{n=0}^{N-1} F(X^\pi(S^n), W^{n+1}), \tag{2.5}$$

where $Z = (W^1, \ldots, W^N)$. This allows us to write (2.4) as

$$\max_\pi \mathbb{E}_Z G(x^{\pi,N}, Z). \tag{2.6}$$

Finally, imagine that we can let our policy π (or $X^\pi(S)$) be represented by a choice of function f, and any parameters θ that are needed to determine the function, which means we can write $y = (f, \theta)$. This means we can rewrite (2.6) as

$$\max_y \mathbb{E}_Z G(x^{y,N}, Z). \tag{2.7}$$

Comparing (2.7) to (2.1), all we need is a slight change of variables to write (2.7) as (2.1).

This little exercise will allow us to reduce virtually all stochastic optimization problems into the same basic form. This observation hides some serious computational issues, which is why we will not be able to solve all problems the same way.

2.1.2 Robust optimization

A variant of the basic stochastic optimization problem is one where we need to make a decision, such as the design of a device or structure, that works under the *worst* possible settings of the uncontrollable parameters. Examples where robust optimization might arise are

■ **EXAMPLE 2.1**

A structural engineer has to design a tall building that minimizes cost (which might involve minimizing materials) so that it can withstand the worst storm conditions in terms of wind speed and direction.

■ **EXAMPLE 2.2**

An engineering designing wings for a large passenger jet wishes to minimize the weight of the wing, but the wing still has to withstand the stresses under the worst possible conditions.

The classical notation used in the robust optimization community is to let u be the uncertain parameters, where we assume that u falls within an *uncertainty set* $\mathcal{U}$. The robust optimization problem is stated as

$$\min_{x \in \mathcal{X}} \max_{u \in \mathcal{U}} F(x, u). \tag{2.8}$$

Creating the uncertainty set $\mathcal{U}$ can be a difficult challenge. For example, if u is a vector with element u_i, one way to formulate $\mathcal{U}$ is the box:

$$\mathcal{U} = \{u | u_i^{lower} \leq u_i \leq u_i^{upper}, \ \forall i\}.$$

The problem is that the worst outcome in $\mathcal{U}$ is likely to be one of the corners of the box, where all the elements u_i are at their upper or lower bound. In practice, this is likely to be an extremely rare event. A more realistic uncertainty set captures the likelihood that a vector u may happen.

2.1.3 The multiarmed bandit problem

The classic information acquisition problem is known as the *bandit problem* which is a colorful name for our cumulative reward problem introduced above. This problem has received considerable attention since it was first introduced in the 1950's, attracting the attention of many hundreds of papers.

Consider the situation faced by a gambler trying to choose which slot machine $x \in \mathcal{X} = \{1, 2, ..., M\}$ to play. Now assume that the winnings may be different for each machine, but the gambler does not know the distributions. The only way to obtain information is to actually play a slot machine. To formulate this problem, let

$$x^n = \begin{cases} 1 & \text{The machine we choose to play next after finishing the } nth \text{ trial,} \\ 0 & \text{otherwise.} \end{cases}$$

$$W_x^n = \text{Winnings from playing slot machine } x = x^{n-1} \text{ during the } n^{th} \text{ trial.}$$

We choose what arm to play in the nth trial after finishing the $n-1st$ trial. We let S^n be the belief state after playing n machines. For example, let

$$\begin{aligned} \mu_x &= \text{A random variable giving the true expected winnings from machine } x, \\ \mu_x^n &= \text{Our estimate of the expected value of } \mu \text{ after } n \text{ trials,} \\ \sigma_x^{2,n} &= \text{The variance of our belief about } \mu_x \text{ after } n \text{ trials.} \end{aligned}$$

Now assume that our belief about μ is normally distributed (after n trials) with mean μ_x^n and variance $\sigma_x^{2,n}$. We can write our belief state as

$$S^n = (\mu_x^n, \sigma_x^{2,n})_{x \in \mathcal{X}}.$$

Our challenge is to find a policy $X^\pi(S^n)$ that determines which machine x^n to play for the $n + 1st$ trial. We have to find a policy that allows us to better learn the true means μ_x, which means we are going to have to sometimes play a machine x^n where the estimated reward μ_x^n is not the highest, but where we acknowledge that this estimate may not be accurate. However, we may end up playing a machine whose average reward μ_x actually is lower than the best, which means we are likely to incur lower winnings. The problem is to find the policy that maximizes winnings over time.

One way to state this problem is to maximize expected discounted winnings over an infinite horizon

$$\max_\pi \mathbb{E} \sum_{n=0}^{\infty} \gamma^n W_{X^\pi(S^n)}^{n+1},$$

where $\gamma < 1$ is a discount factor. Of course, we could also pose this as a finite horizon problem (with or without discounting).

An example of a policy that does quite well is known as the interval estimation policy, given by

$$X^{IE,n}(S^n|\theta^{IE}) = \arg\max_{x \in \mathcal{X}} \left(\mu_x^n + \theta^{IE} \bar{\sigma}_x^{2,n} \right),$$

where $\bar{\sigma}_x^{2,n}$ is our estimate of the variance of μ_x^n, given by

$$\bar{\sigma}_x^{2,n} = \frac{\sigma_x^{2,n}}{N_x^n}.$$

This is parameterized by θ^{IE} which determines how much weight to put on the uncertainty in the estimate μ_x^n. If $\theta^{IE} = 0$, then we have a pure exploitation policy where we are simply choosing the alternative that seems best. As θ^{IE} increases, we put more emphasis on the uncertainty in the estimate. As we are going to see in chapter 7, effective learning policies have to strike a balance between exploring (trying alternatives which are uncertain) and exploiting (doing what appears to be best).

The multiarmed bandit problem is an example of an online learning problem (that is, where we have to learn by doing), where we want to maximize the cumulative rewards. Some examples of these problems are

---

■ **EXAMPLE 2.1**

Consider someone who has just moved to a new city and who now has to find the best path to work. Let T_p be a random variable giving the time he will experience if he chooses path p from a predefined set of paths $\mathcal{P}$. The only way he can obtain observations of the travel time is to actually travel the path. Of course, he would like to choose the path with the shortest average time, but it may be necessary to try a longer path because it may be that he simply has a poor estimate. The problem is identical to our bandit problem if we assume that driving one path does not teach us anything about a different path (this is a richer form of bandit problem).

■ **EXAMPLE 2.2**

A baseball manager is trying to decide which of four players makes the best designated hitter. The only way to estimate how well they hit is to put them in the batting order as the designated hitter.

■ **EXAMPLE 2.3**

A doctor is trying to determine the best blood pressure medication for a patient. Each patient responds differently to each medication, so it is necessary to try a particular medication for a while, and then switch if the doctor feels that better results can be achieved with a different medication.

Multiarmed bandit problems have long history as a niche problem in applied probability (going back to the 1950's) and computer science (starting in the mid 1980's). The bandit community (the researchers who use the vocabulary of "bandits" (for a problem) and "arms" (for an alternative) has broadened to consider a much wider range of problems. We revisit this important problem class in chapter 7.

2.1.4 Decision trees

Decision trees are easily one of the most familiar ways to depict sequential decision problems, with or without uncertainty. Figure 2.1 illustrates a simple problem of determining whether to hold or sell an asset. If we decide to hold, we observe changes in the price of the asset and then get to make the decision of holding or selling.

Figure 2.1 illustrates the basic elements of a decision tree. Square nodes represent points where decisions are made, while circles represent points where random information is revealed. We solve the decision tree by rolling backward, calculating the value of being at each node. At an outcome node, we average across all the downstream nodes (since we do not control which node we transition to), while at decision nodes, we pick the best decision based on the one-period reward plus the downstream value.

Almost any dynamic program with discrete states and actions can be modeled as a decision tree. The problem is that they are not practical when there are a large number of states, actions and random outcomes, since the tree grows extremely quickly. The breakthrough known as "dynamic programming" was the recognition that there were many applications where the number of states was not that large, and this could be exploited to keep the decision tree from exploding. The inventory/storage problems considered next are an example of this problem class.

2.1.5 Online computation

There is an entire field of research known as *online computation* which was originally motivated by the need to make decisions very quickly, with limited computational resources. Such situations arise, for example, when a thermostat has to make a decision about whether to turn an air conditioner on or off, or how to respond to a military situation in the field on mobile devices. However, this is also precisely the setting that arises today when Uber assigns drivers to customers, or when Google has to decide what ad to display next to search results.

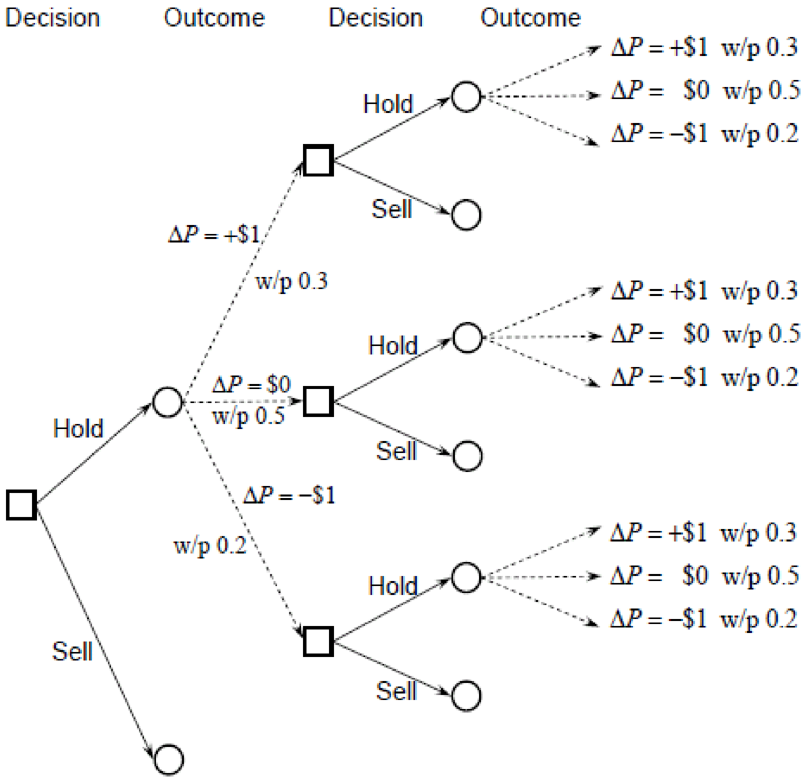


Figure 2.1 Decision tree illustrating the sequence of decisions (hold or sell an asset) and new information (price changes).

The setting of online computation is typically described as one where there is no information about the environment, with no access to a forecast or a model of future events. Decisions might be made using simple rules (for example, turn the air conditioner on if the temperature is greater than 72 degrees F, and turn it off when it falls to 70 degrees F). Alternatively, we might look at a section of actions (such as picking Uber drivers to serve a customer) where we want to pick the best action. Let $C(S_t, x)$ be the cost of choosing driver x in a set $\mathcal{X}_t$ (the eight closest drivers), which depends on the state S_t. We might choose the best driver by solving

$$x_t = \arg\min_{x \in \mathcal{X}_t} C(S_t, x).$$

The online computation community will insist that they cannot make decisions that consider the impact on the future because they do not know anything about the future. The research community likes to prove bounds on the regret, which quantifies how much better you might do with perfect information about the future.

2.1.6 Two-stage stochastic programming

Imagine that we have a problem where we have to decide how many Christmas trees to plant which will determine our inventory five years into the future. Call this decision x_0 (we suppress the year we are planning for), which may be a vector determining the plantings at different wholesalers around a region. Then, when we arrive at the sales year (five years later), we see the demand D_1 for Christmas trees and the prices p_1 that the retailers are willing to pay and then have to make shipping decisions x_1 to each retailer. Let $W_1 = (D_1, p_1)$ represent this random information, and let ω refer to a sample realization of W_1, so that $W_1(\omega) = (D_1(\omega), p_1(\omega))$ is one possible realization of demands and prices. We make the decision x_1 after we see this information, so we have a decision $x_1(\omega)$ for each possible realization. Assume that c_0 is the costs of producing trees at each location, and c_1 are the costs associated with the vector x_1.

Assume for the moment that $\Omega = (\omega_1, \omega_2, \ldots, \omega_K)$ is a (not too large) set of possible outcomes for the demand $D_1(\omega)$ and price $p_1(\omega)$. Our second stage decisions $x_1(\omega)$ are constrained by what we planted in the first stage x_0, and we are limited by how much we sell. These two constraints are written as

$$\begin{aligned} A_1 x_1(\omega) &\leq x_0, \\ B_1 x_1(\omega) &\leq D_1(\omega). \end{aligned}$$

Let $\mathcal{X}_1(\omega)$ be the feasible region for $x_1(\omega)$ defined by these constraints. This allows us to write our problem over both stages as

$$\max_{x_0} \left(-c_0 x_0 + \sum_{\omega \in \Omega} p(\omega) \max_{x_1(\omega) \in \mathcal{X}_t(\omega)} \big((p_1(\omega) - c_1) x_1(\omega) \big) \right). \tag{2.9}$$

In the language of stochastic programming, the second stage decision variables, $x_1(\omega)$, are called "recourse variables" since they represent how we may respond as new information becomes available (which is the definition of "recourse"). Two-stage stochastic programs are basically deterministic optimization problems, but they can be *very large* deterministic optimization problems, albeit ones with special structure.

For example, imagine that we allow the first stage decision x_0 to "see" the information in the second stage, in which case we would write it as $x_0(\omega)$. In this case, we obtain a series of smaller problems, one for each ω. However, now we are allowing x_0 to cheat by seeing into the future. We can overcome this by introducing a *nonanticipativity constraint* which might be written

$$x^0(\omega) - x^0 = 0.$$

Now, we have a family of first stage variables $x_0(\omega)$, one for each ω, and then a single variable x_0, where we are trying to force each $x_0(\omega)$ to be the same (at which point we would say that x_0 is "nonanticipative"). Algorithmic specialists can exploit the nonanticipacity constraint by relaxing it, then solving a series of smaller problems (perhaps in parallel), and then introducing linking mechanisms so that the overall procedure converges toward a solution that satisfies the nonanticipativity constraint.

2.1.7 Chance constrained problems

There are problems where we have to satisfy a constraint that depends on uncertain information at the time we make a decision. For example, we may wish to allocate inventory with the goal that we cover demand 80 percent of the time. Alternatively, we may wish to schedule a flight so that it is on time 90 percent of the time. We can state these problems using the general form

$$\min_x f(x), \tag{2.10}$$

subject to the probabilistic constraint (often referred to as a chance constraint)

$$\mathbb{P}[C(x, W) \geq 0] \quad \leq \quad \alpha, \tag{2.11}$$

where $0 \leq \alpha \leq 1$. The constraint (2.11) is often written in the equivalent form

$$\mathbb{P}[C(x, W) \leq 0] \quad \geq \quad 1 - \alpha. \tag{2.12}$$

Here, $C(x, W)$ is the amount that a constraint is violated (if positive). Using our examples, it might be the demand minus the inventory which is the lost demand if positive, or the covered demand if negative. Or, it could be the arrival time of a plane minus the scheduled time, where positive means a late arrival.

Chance constrained programming is a method for handling a particular class of constraints that involve uncertainty, typically in the setting of a static problem: make decision, see information, stop. Chance constrained programs convert these problems into deterministic, nonlinear programs, with the challenge of computing the probabilistic constraint within the search algorithm.

2.1.8 Optimal stopping

A classical problem in stochastic optimization is known as the optimal stopping problem. Imagine that we have a stochastic process W_t (this might be prices of an asset) which determines a reward $f(W_t)$ if we stop at time t (the price we receive if we stop and sell the asset). Let $\omega \in \Omega$ be a sample path of $W_1, \ldots, W_T$ (we are going to limit our discussion to finite horizon problems, which might represent a maturation date on a financial option). Let

$$X_t(\omega) = \begin{cases} 1 & \text{If we stop at time } t, \\ 0 & \text{Otherwise.} \end{cases}$$

Let τ be the time t when $X_t = 1$ (we assume that $X_t = 0$ for $t > \tau$). This notation creates a problem, because ω specifies the entire sample path, which seems to suggest that we are allowed to look into the future before making our decision at time t (don't laugh - this mistake is not just easy to make, it is actually a fairly standard approximation in the field of stochastic programming which we revisit in chapter 20).

To fix this, we require that the function X_t be constructed so that it depends only on the history $W_1, \ldots, W_t$. When this is the case τ is called a *stopping time*. The optimization problem can then be stated as

$$\max_\tau \mathbb{E} X_\tau f(W_\tau), \tag{2.13}$$

where we require τ to be a "stopping time." Mathematicians will often express this by requiring that τ (or equivalently, X_t) be an "$\mathcal{F}_t$-measurable function." This language is familiar to students with training in measure-theoretic probability, which is *not* necessary for developing models and algorithms for stochastic optimization. Later, we are going to provide an easy introduction to these ideas (in chapter 9, and then explain why we do not need to use this vocabulary.

More practically, the way we are going to solve the stopping problem in (2.13) is that we are going to create a function $X^\pi(S_t)$ that depends on the state of the system at time t. For example, imagine that we need a policy for selling an asset. Let $R_t = 1$ if we are holding the asset, and 0 otherwise. Assume that $p_1, p_2, \ldots, p_t$ is the history of the price process, where we receive p_t if we sell at time t. Further assume that we create a smoothed process $\bar{p}_t$ using

$$\bar{p}_t = (1-\alpha)\bar{p}_{t-1} + \alpha p_t.$$

At time t, our state variable is $S_t = (R_t, \bar{p}_t, p_t)$. A sell policy might look like

$$X^\pi(S_t|\theta) = \begin{cases} 1 & \text{If } \bar{p}_t > \theta^{max} \text{ or } \bar{p}_t < \theta^{min}, \\ 0 & \text{Otherwise.} \end{cases}$$

Finding the best policy means finding the best $\theta = (\theta^{min}, \theta^{max})$ by solving

$$\max_\theta \mathbb{E} \sum_{t=0}^{T} p_t X^\pi(S_t|\theta).$$

Our stopping time, then, is the earliest time $\tau = t$ where $X^\pi(S_t|\theta) = 1$.

Optimal stopping problems arise in a variety of settings. Some examples include:

American options - An American option on a financial asset gives you the right to sell the asset at the current price on or before a specified date.

Machine replacement - While monitoring the status of a (typically complex) piece of machinery, we need to create a policy that tells us when to stop and repair or replace.

Homeland security - The National Security Administration collects information on many people. The NSA needs to determine when to start tracking someone, when to stop (if they feel the target is of no risk) or when to act (when they feel the target is of high risk).

Health intervention - Public health officials are continually tracking the presence of disease in a population. They need to make the call when a pattern of occurrences constitutes an actionable outbreak.

Optimal stopping may look like a disarmingly easy problem, given the simplicity of the state variable. However, in real applications there is almost always additional information that needs to be considered. For example, our asset selling problem may depend on a basket of indices or securities that greatly expands the dimensionality of the state variable. The machine replacement problem might involve a number of measurements that are combined to make a decision. The homeland security application could easily involve a number of factors (places the person has visited, the nature of communications, and recent purchases). Finally, health decisions invariably depend on a number of factors that are unique to each patient.

2.1.9 Markov decision processes

Assume that our system is described a set of discrete states $s \in \mathcal{S} = \{1, 2, \ldots, |\mathcal{S}|\}$, and discrete actions $a \in \mathcal{A}$ (which might depend on the state that we are in, in which case we might write $\mathcal{A}_s$). Further assume we receive a reward $r(s, a)$ if we take action a while in state s. Finally (and this is the strongest assumption) assume that we are given a set of transition probabilities

$P(s'|s, a) =$ The probability that state $S_{t+1} = s'$ given that we are in state $S_t = s$ and take action a.

If we are solving a finite horizon problem, let $V_t(S_t)$ be the optimal value of being in state S_t and behaving optimally from time t onward. If we are given $V_{t+1}(S_{t+1})$, we can compute $V_t(S_t)$ using

$$V_t(S_t) = \max_{a \in \mathcal{A}_s} \big(r(S_t, a) + \sum_{s' \in \mathcal{S}} P(s'|S_t, a) V_{t+1}(s') \big). \tag{2.14}$$

Equation (2.14) may seem somewhat obvious, but when first introduced it was actually quite a breakthrough, and is known as Bellman's optimality equation in operations research and computer science, or Hamilton-Jacobi equations in control theory (although this community typically writes it for deterministic problems).

Equation (2.14) is the foundation for a major class of policies that we refer to as policies based on value function (or VFA policies), recognizing that if we can actually compute (2.14), then the value function is optimal, giving us a rare instance of an optimal policy.

If the one-step transition matrix $P(s'|S_t, a)$ can be computed (and stored), then equation (2.14) is quite easy to compute starting at time T (when we assume $V_T(S_T)$ is given, where it is fairly common to use $V_T(S_T) = 0$). Perhaps for this reason, there has been considerable interest in this community on steady state problems, where we assume that as $t \rightarrow \infty$, that $V_t(S_t) \rightarrow V(S)$. In this case, (2.14) becomes

$$V(s) = \max_{a \in \mathcal{A}_s} \left(r(s, a) + \sum_{s' \in \mathcal{S}} P(s'|s, a) V(s') \right). \tag{2.15}$$

Now we have a system of equations that we have to solve to find $V(s)$. We review these methods in some depth in chapter 14.

Bellman's equation was viewed as a major computational breakthrough when it was first introduced, because it avoids the explosion of decision trees. However, people (including Bellman) quickly realized realized that there was a problem when the state s is a vector (even if it is still discrete). The size of the state space grows exponentially with the number of dimensions, typically limiting this method to problems where the state variable has at most three or four dimensions.

Bellman's equation actually suffers from three curses of dimensionality. In addition to the state variable, the random information (buried in the one-step transition $P(s'|s, a)$) might also be a vector. Finally, the action a might be a vector x. It is common for people to dismiss "dynamic programming" (but they mean discrete Markov decision processes) because of "the curse of dimensionality" (they could say because of "the curses of dimensionality"), but the real issue is the use of lookup tables. There are strategies for overcoming the curses of dimensionality, but if it were easy, this would be a much shorter book.

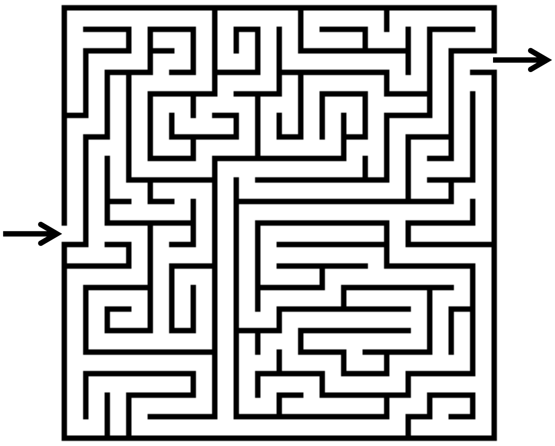


Figure 2.2 Finding a path through a maze.

2.1.10 Reinforcement learning

Reinforcement learning evolved out of efforts to model animal behavior, as with mice trying to find their way through a maze to a reward (see figure 2.2). Successes were learned over time by capturing the probability that a path from a particular point in the maze eventually leads to a success. This work was started in the early 1980's by Rich Sutton and his academic adviser Andy Barto. At some point, people realized that the central ideas could all be cast in the language of Markov decision processes, but without the use of backward dynamic programming that are implied in the calculation of equation (2.14) above. Instead of stepping backward in time, reinforcement learning proceeds by stepping forward in time, but then performing backward updates. These ideas became the foundation of what later became known as approximate (or adaptive) dynamic programming. The process of stepping forward avoids the need to loop over all possible states, as is required if solving equation (2.15).

Despite the similarities between reinforcement learning and what we will later introduce as approximate dynamic programming, the reinforcement learning community had its roots in a particular class of problems. For example, the prototypical reinforcement learning problem is "model free" which means that we do not have access to the one-step transition matrix $P(s'|s,a)$ which is considered a fundamental input to a Markov decision process. Instead, we assume that if we are in a state S_t and take an action a_t, that we would then observe S_{t+1} without having access to the equations that describe how we get there.

Instead of learning the value $V(s)$ of being in a state s, the core algorithmic strategy of reinforcement learning involves learning the value $Q(s,a)$ of being in a state s and then taking an action a. The basic algorithm, known as Q-learning, proceedings by computing

$$\hat{q}^n(s^n,a^n) = r(s^n,a^n) + \lambda \max_{a'} \bar{Q}^{n-1}(s',a'), \tag{2.16}$$

$$\bar{Q}^n(s^n,a^n) = (1-\alpha_{n-1})\bar{Q}^{n-1}(s^n,a^n) + \alpha_{n-1}\hat{q}^n(s^n,a^n). \tag{2.17}$$

To compute (2.16), we assume we are given a state s^n. We then use some method to choose an action a^n, which produces a reward $r(s^n,a^n)$. We then randomly sample a downstream state s' that might result from being in a state s^n and taking action a^n. Using our simulated downstream state s', we then find what appears to be the best action a' based on our current estimates $\bar{Q}^{n-1}(s', a')$ (known as ”Q-factors”). We then update the estimates of the value of being in states s^n and action a^n. When this logic is applied to our maze in figure 2.2, the algorithm steadily learns the state/action pairs with the highest probability of finding the exit, but it does require sampling all states and actions often enough.

There are many variations of Q-learning that reflect different rules for choosing the state s^n, choosing the action a^n, what is done with the updated estimate $\hat{q}^n(s^n, a^n)$, and how the estimates $\bar{Q}^n(s, a)$ are calculated (equations (2.16) - (2.17) reflect a lookup table representation). These basic equations would only work for problems with relatively small states and actions, which means we could simply use equation (2.14) if we know the one-step transition matrix (hence we see why this community typically assumes that we do not know the transition matrix).

2.1.11 Optimal control

The optimal control community is most familiar with the deterministic form of a control problem, which is typically written in terms of the “system model” (transition function)

$$x_{t+1} = f(x_t, u_t),$$

where x_t is the state variable and u_t is the control. The problem is to find u_t that solves

$$\min_u \sum_{t=0}^{T} L(x_t, u_t) + J_T(x_T), \tag{2.18}$$

where $L(x, u)$ is a “loss function” and $J_T(x_T)$ is a terminal cost (the notation here is not standard).

A solution strategy which is so standard that it is often stated as part of the model is to view the transition $x_{t+1} = f(x_t, u_t)$ as a constraint that can be relaxed, producing the objective

$$\min_u \sum_{t=0}^{T} \big(L(x_t, u_t) + \lambda_t(x_{t+1} - f(x_t, u_t))\big) + J_T(x_T), \tag{2.19}$$

where λ_t is a set of Lagrange multipliers known as “co-state variables.” The function

$$H(x_0, u) = \sum_{t=0}^{T} \big(L(x_t, u_t) + \lambda_t(x_{t+1} - f(x_t, u_t))\big) + J_T(x_T)$$

is known as the Hamiltonian.

The most common stochastic version would be written

$$x_{t+1} = f(x_t, u_t, w_t)$$

where w_t is random at time t. Often, w_t is viewed as a form of additive noise that arises when trying to measure the state of the system, which consists of the location and speed of an aircraft. In this setting, the noise is additive, and would be written

$$x_{t+1} = f(x_t, u_t) + w_t. \tag{2.20}$$

When we introduce noise, we can still write the objective as (2.24), but now we have to follow the objective function with the statement "where u_t is $\mathcal{F}_t$-measurable" (see section 9.12.2 for an explanation of this term). The problem with this formulation is that it does not provide an immediate path to computation. As readers will see, the way this problem is solved is to construct a policy, that we denote (using the notation of this example) by $U^\pi(x_t)$, which means the function has to determine a decision $u_t = U^\pi(x_t)$ using only the information in the state x_t. The search for solution that satisfies the requirement that "u_t be $\mathcal{F}_t$-measurable" is the same as finding some function that depends on x_t (which in turn can only depend on information that has arrived before time t). A significant part of this book is focused on describing methods for finding these functions.

A common form for the objective in (2.24) is an objective function that is quadratic in the state x_t and control u_t, given by

$$\min_\pi \mathbb{E} \sum_{t=0}^{T} \big((x_t)^T Q_t x_t + (u_t)^T R_t u_t\big). \tag{2.21}$$

Although it takes quite a bit of algebra, it is possible to show that the optimization problem in (2.24), with either a deterministic transition or stochastic with additive noise as in (2.20), has the form

$$U^*(x_t) = K_t x_t, \tag{2.22}$$

where K_t is a suitably dimensioned matrix that depends on the matrices $(Q_{t'}, R_{t'}), t' \leq t$.

2.1.12 Model predictive control

A popular way of solving control problems is to come up with a function that depends only on the current state to help find the best action now. One example of this is our optimal control policy we just presented in equation (2.22) for optimal control problems. Another example is use of Q-factors from reinforcement learning which we calculated in equations (2.16) - (2.17), which gives us the policy

$$A^\pi(S_t) = \arg\max_a \bar{Q}^n(S_t, a). \tag{2.23}$$

These are sometimes known as "model-free" policies because they do not require a model of the physical problem to determine an action.

There are, however, many settings where we need to think about what is going to happen in the future. The most common form in the optimal control literature is known as *model predictive control* which simply means choosing a decision now by optimizing over some horizon, which we can write as

$$U^\pi(x_t) = \arg\min_{u_t} \left(L(x_t, u_t) + min_{u_{t+1},\ldots,u_{t+H}} \sum_{t'=t}^{t+H} L(x_{t'}, u_{t'}) \right). \tag{2.24}$$

The optimization problem in (2.24) requires a model over the horizon $t, \ldots, t+H$, which means we need to be able to model losses as well as the system dynamics using $x_{t+1} = f(x_t, u_t)$. A slightly more precise name for this might be "model-based predictive control", but "model predictive control" (or MPC, as it is often known) is the term that evolved in the controls community.

Model predictive control is most often written using a deterministic model of the future, primarily because most control problems are deterministic. However, a common approximation for stochastic problems is still to use a deterministic approximation of the future. This is so common that model predictive control is often interpreted as a deterministic model of the future, even when the problem being solved (such as (2.21)) is stochastic. However, the proper use of the term refers to *any* model of the future (even an approximation) that is used to make a decision now.

Model predictive control is a widely used idea, often under names such as "rolling horizon heuristic" or "receding horizon heuristic." When you use a navigation system to plan the shortest path to a destination, that is a form of model predictive control where a deterministic model of travel times is used to determine where to turn next. In fact, a decision tree can be viewed as a form of model predictive control when it is used in a fully sequential setting.

2.2 A SIMPLE MODELING FRAMEWORK FOR SEQUENTIAL DECISION PROBLEMS

Now that we have covered a number of simple examples, it is useful to briefly review the elements of a dynamic program. We are going to revisit this topic in considerably greater depth in chapter 9, but this discussion provides a brief introduction. Our presentation focuses on *stochastic* dynamic programs which exhibit a flow of uncertain information. These problems, at a minimum, consist of the following elements:

The state variable - S_t This captures all the information we need to model the system from time t onward, which means computing the cost/contribution function, constraints on decisions, and any other variables needed to model the transition of this information over time. The state S_t may consist of the physical resources R_t (such as inventories), other information I_t (price of a product, weather), and the belief state B_t which captures information about a probability distribution describing uncertain variables or parameters. It is important to recognize that the state variable, regardless of whether it is describing physical resources, attributes of a system, or the parameters of a probability distribution, is always a form of information.

The decision variable - x_t **/** a_t **/** u_t Decisions/actions/controls represent how we control the process. Decisions (or actions or controls) are determined by decision functions known as *policies*, also known as control laws in control theory.

Exogenous information - W_t This is the information that first becomes known at time t from an exogenous source (for example, the demand for product, the speed of the wind, the outcome of a medical treatment, the results of a laboratory experiment).

The transition function - This function determines how the system evolves from the state S_t to the state S_{t+1} given the decision that was made at time t and the new information that arrived between t and $t + 1$.

The objective function - This function specifies the costs being minimized, the contributions/rewards being maximized, or other performance metrics. The objective is to find the

We now illustrate this framework using an asset acquisition problem.

The state variable is the information we need to make a decision and compute functions that determine how the system evolves into the future. In our asset acquisition problem, we need three pieces of information. The first is R_t, the resources on hand before we make any decisions (including how much of the demand to satisfy). The second is the demand itself, denoted D_t, and the third is the price p_t. We would write our state variable as $S_t = (R_t, D_t, p_t)$.

We have two decisions to make. The first, denoted x_t^D, is how much of the demand D_t during time interval t that should be satisfied using available assets, which means that we require $x_t^D \leq R_t$. The second, denoted x_t^O, is how many new assets should be acquired at time t which can be used to satisfy demands during time interval $t+1$.

The exogenous information process consists of three types of information. The first is the new demands that arise during time interval t, denoted $\hat{D}_t$. The second is the change in the price at which we can sell our assets, denoted $\hat{p}_t$. Finally, we are going to assume that there may be exogenous changes to our available resources. These might be blood donations or cash deposits (producing positive changes), or equipment failures and cash withdrawals (producing negative changes). We denote these changes by $\hat{R}_t$. We often use a generic variable W_t to represent all the new information that is first learned during time interval t, which for our problem would be written $W_t = (\hat{R}_t, \hat{D}_t, \hat{p}_t)$. In addition to specifying the types of exogenous information, for stochastic models we also have to specify the likelihood of a particular outcome. This might come in the form of an assumed probability distribution for $\hat{R}_t$, $\hat{D}_t$, and $\hat{p}_t$, or we may depend on an exogenous source for sample realizations (the actual price of the stock or the actual travel time on a path).

Once we have determined what action we are going to take from our decision rule, we compute our contribution $C_t(S_t, x_t)$ which might depend on our current state and the action x_t that we take at time t. For our asset acquisition problem (where the state variable is R_t), the contribution function is

$$C_t(S_t, x_t) = p_t x_t^D - c_t x_t^O.$$

In this particular model, $C_t(S_t, x_t)$ is a deterministic function of the state and action. In other applications, the contribution from action x_t depends on what happens during time $t+1$.

Next, we have to specify how the state variable changes over time. This is done using a *transition function* which we might represent in a generic way using

$$S_{t+1} = S^M(S_t, x_t, W_{t+1}),$$

where S_t is the state at time t, x_t is the decision we made at time t and W_{t+1} is our generic notation for the information that arrives between t and $t+1$. We use the notation $S^M(\cdot)$ to denote the transition function, where the superscript M stands for "model" (or "system model" in recognition of vocabulary that has been in place for many years in the engineering community). The transition function for our asset acquisition problem is given by

$$\begin{aligned} R_{t+1} &= R_t - x_t^D + x_t^O + \hat{R}_{t+1}, \\ D_{t+1} &= D_t - x_t^D + \hat{D}_{t+1}, \\ p_{t+1} &= p_t + \hat{p}_{t+1}. \end{aligned}$$

This model assumes that unsatisfied demands are held until the next time period.

Our final step in formulating a dynamic program is to specify the objective function. Assume we are trying to maximize the total contribution received over a finite horizon

$t = (0, 1, \ldots, T)$. If we were solving a deterministic problem, we might formulate the objective function as

$$\max_{(x_t)_{t=0}^T} \sum_{t=0}^{T} C_t(S_t, x_t). \tag{2.25}$$

We would have to optimize (2.25) subject to a variety of constraints on the actions $(x_0, x_1, \ldots, x_T)$.

If we have a stochastic problem, which is to say that there are a number of possible realizations of the exogenous information process $(W_t)_{t=1}^T$, then we have to formulate the objective function in a different way. If the exogenous information process is uncertain, we do not know which state we will be in at time t. Since the state S_t is a random variable, then the decision (which depends on the state) is also a random variable.

We get around this problem by formulating the objective in terms of finding the best *policy* (or decision rule) for choosing decisions. A policy tells us what to do for all possible states, so regardless of which state we find ourselves in at some time t, the policy will tell us what decision to make. This policy must be chosen to produce the best *expected* contribution over all outcomes. If we let $X^\pi(S_t)$ be a particular decision rule indexed by π, and let Π be a set of decision rules, then the problem of finding the best policy would be written

$$\max_{\pi \in \Pi} \mathbb{E} \sum_{t=0}^{T} C_t(S_t, X^\pi(S_t)). \tag{2.26}$$

Exactly what is meant by finding the best policy out of a set of policies is very problem specific. Our decision rule might be to order $X^\pi(R_t) = S - R_t$ if $R_t < s$ and order $X^\pi(R_t) = 0$ if $R_t \geq s$. The family of policies is the set of all values of the parameters (s, S) for $s < S$ (here, s and S are parameters to be determined, not state variables). If we are selling an asset, we might adopt a policy of selling if the price of the asset p_t falls below some value $\bar{p}$. The set of all policies is the set of all values of $\bar{p}$. However, policies of this sort tend to work only for very special problems.

Equation (2.26) states our problem as one of finding the best policy (or decision rule, or function) X^π to maximize the expected value of the total contribution over our horizon. There are a number of variations of this objective function. For applications where the horizon is long enough to affect the time value of money, we might introduce a discount factor γ and solve

$$\max_{\pi \in \Pi} \mathbb{E} \sum_{t=0}^{T} \gamma^t C_t(S_t, X^\pi(S_t)), \tag{2.27}$$

where γ is a discount factor where $0 \leq \gamma \geq 1$. There is also considerable interest in infinite horizon problems of the form

$$\max_{\pi \in \Pi} \mathbb{E} \sum_{t=0}^{\infty} \gamma^t C_t(S_t, X^\pi(S_t)). \tag{2.28}$$

Equation (2.28) is often used when we want to study the behavior of a system in steady state.

Equations such as (2.26), (2.27), and (2.28) are all easy to write on a sheet of paper. Solving them computationally is a different matter. That challenge is the focus of this book.

This description provides only a taste of the richness of sequential decision processes. Chapter 9 describes the different elements of a dynamic program in far greater detail.

2.3 APPLICATIONS

We now illustrate our modeling framework using a series of applications. These problems illustrate some of the modeling issues that can arise. We often start from a simpler problem, and then show how details can be added. Pay attention to the growth in the dimensionality of the state variable as these complications are introduced.

2.3.1 The newsvendor problems

A popular problem in operations research is known as the newsvendor problem, which is described as the story of deciding how many newspapers to put out for sale to meet an unknown demand. The newsvendor problem arises in many settings where we have to choose a fixed parameter that is then evaluated in a stochastic setting. It often arises as a subproblem in a wide range of resource allocation problems (managing blood inventories, budgeting for emergencies, allocating fleets of vehicles, hiring people), it also arises in other settings, such as bidding a price for a contract (bidding too high means you may lose the contract), or allowing extra time for a trip.

2.3.1.1 Basic newsvendor - Terminal reward The basic newsvendor is modeled as

$$F(x, W) = p \min\{x, W\} - cx, \tag{2.29}$$

where x is the number of newspapers we have to order before observing our random "demand" W. We sell our newspapers at a price p (the smaller of x and W), but we have to buy them at a unit cost c. The goal is to solve the problem

$$\max_x \mathbb{E}_W F(x, W). \tag{2.30}$$

A special case of the newsvendor problem is when the distribution of W is known (see exercise 2.1). In most cases, the newsvendor problem arises in settings where we can observe W, but we do not known its distribution (this is often referred to as *data driven*). In this setting, we assume that we have to determine the amount to order x^n at the end of day n, after which we observe demand W^{n+1}, giving us a profit (at the end of day $n+1$) of

$$\hat{F}^{n+1} = F(x^n, W^{n+1}) = p \min\{x^n, W^{n+1}\} - cx^n.$$

After each iteration, we may assume we observe W^{n+1}, although often we only observe $\min(x^n, W^{n+1})$ (which is known as censored observations). We can devise strategies to try to learn the distribution of W, and then use our ability to solve the problem optimally (given in exercise 2.1).

Another approach is to try to learn the function $\mathbb{E}_w F(x, W)$ directly. Either way, let S^n be our belief state (about W, or about $\mathbb{E}_w F(x, W)$) about our unknown quantities. S^n might be a point estimate, but it is often a probability distribution. For example, we might let $\mu_x = \mathbb{E}F(x, W)$ where we assume that x is discrete (say, the number of newspapers). After n iterations, we might have estimates $\bar{\mu}_x^n$ of $\mathbb{E}F(x, W)$, with standard deviation $\bar{\sigma}_x^n$ where we would then assume that $\mu_x \sim N(\bar{\mu}_x^n, \bar{\sigma}_x^{n,2})$. In this case, we would write $S^n = (\bar{\mu}^n, \bar{\sigma}^n)$ where $\bar{\mu}^n$ and $\bar{\sigma}^n$ are both vectors over all values of x.

Given our (belief) state S^n, we then have to define a policy (we might also call this a rule, or it might be a form of algorithm) that we denote by $X^\pi(S^n)$ where $x^n = X^\pi(S^n)$ is the decision we are going to use in our next trial where we either observe W^{n+1} or $\hat{F}^{n+1}$. While we would like to run this policy until $n \to \infty$, in practice we are going to be limited to N trials which then gives us a solution $x^{\pi,N}$. This solution depends on our initial state S^0, the observations $W^1, \ldots, W^N$ which occurred while we were finding $x^{\pi,N}$, and then we observe W one more time to evaluate $x^{\pi,N}$. We want to find the policy that solves

$$\max_\pi \mathbb{E}_{S_0}\mathbb{E}_{W^1,\ldots,W^N|S^0}\mathbb{E}_W F(x^{\pi,N}, W). \tag{2.31}$$

2.3.1.2 Basic newsvendor - cumulative reward A more realistic presentation of an actual newsvendor problem recognizes that we are accumulating profits while simultaneously learning about the demand W (or the function $\mathbb{E}_W F(x,W)$). If this is the case, then we would want to find a policy that solves

$$\max_\pi \mathbb{E}_{S_0}\mathbb{E}_{W_1,\ldots,W_T} \sum_{t=0}^{T-1} F(X^\pi(S_t), W_{t+1}). \tag{2.32}$$

2.3.1.3 Contextual newsvendor Imagine a newsvendor problem where the price p of our product is dynamic, given by p_t, which is revealed before we have to make a decisions. Our profits would be given by

$$F(x, W|S_t) = p_t \min\{x, W\} - cx. \tag{2.33}$$

As before, assume that we do not know the distribution of W, and let B_t our belief state about W (or about $\mathbb{E}F(x,W)$). Our state $S_t = (p_t, B_t)$, since we have to capture both the price p_t and our state of belief B_t. We can write our problem now as

$$\max_x \mathbb{E}_W F(x, W|p_t).$$

Now, instead of finding the optimal order quantity x^*, we have to find the optimal order quantity as a function of the price p_t, which we might write as $x^*(p_t)$. While x^* is a deterministic value, $x^*(p)$ is a function.

One solution might be to fix p and then solve a fixed newsvendor problem, but in reality the problem is typically not this simple. We often do not have the distribution of W_t, and depend on an exogenous source (such as the sales of our newspapers) for samples of W_t. This same process may also be driving the evolution of S_t, which in turn might affect the distribution of W_t. For example, imagine an event has happened (such as weather problems creating a shortage), which means that a high value of p_t implies a lower distribution of demand. This means that rather than adaptively learning the optimal solution x^*, we need to adaptively learn the function $x^*(p_t)$.

2.3.1.4 Multidimensional newsvendor problems Newsvendor problems can be multidimensional. One version is the *additive newsvendor problem* where there are K products to serve K demands, but using a production process that limits the total amount delivered. This would be formulated as

$$F(x_1, \ldots, x_K) = E_{W_1,\ldots,W_K} \sum_{k=1}^{K} p_k \min(x_k, W_k) - c_k x_k, \tag{2.34}$$

where

$$\sum_{k=1}^{K} x_k \leq U. \tag{2.35}$$

A second version arises when there are multiple products (different types/colors of cars) trying to satisfy the same demand W. This is given by

$$F(x_1,\ldots,x_K) = \mathbb{E}_W \left\{ \sum_{k=1}^{K} p_k \min\left[x_k, \left(W - \sum_{\ell=1}^{k-1} x_\ell \right)^+ \right] - \sum_{k=1}^{K} c_k x_k \right\}, \tag{2.36}$$

where $(Z)^+ = \max(0, Z)$.

2.3.2 Inventory/storage problems

Inventory (or storage) problems represent an astonishingly broad class of applications that span any problem where we buy/acquire (or sell) a resource to meet a demand, where excess inventory can be held to the next time period. Elementary inventory problems (with discrete quantities) appear to be the first problem to illustrate the power of a compact state space, which overcomes the exponential explosion that occurs if you try to formulate and solve these problems as decision trees.

2.3.2.1 Inventory without lags The simplest problem allows us to order new product x_t at time t that arrives right away.

R_t = Amount of inventory left over at the end of period t,

x_t = Amount ordered at the end of period t that will be available at the beginning of time period $t+1$,

$\hat{D}_{t+1}$ = Demand for the product that arises between t and $t+1$.

c_t = The unit cost of order product for product ordered at time t,

p_t = The price we are paid when we sell a unit during the period $(t, t+1)$.

Our basic inventory process is given by

$$R_{t+1} = \max\{0, R_t + x_t - \hat{D}_{t+1}\}.$$

We add up our total contribution at the end of each period. Let y_t be the sales during time period $(t-1, t)$. Our sales are limited by the demand $\hat{D}_t$ as well as our available product $R_{t-1} + x_{t-1}$, but we are going to allow ourselves to choose how much to sell, which may be smaller than either of these. So we would write

$$\begin{aligned} y_t &\leq R_{t-1} + x_{t-1}, \\ y_t &\leq \hat{D}_t. \end{aligned}$$

We are going to assume that we determine y_t at time t after we have learned the demands D_t for the preceding time period. So, at time t, the revenues and costs are given by

$$C_t(x_t, y_t) \quad = \quad p_t y_t - c_t x_t.$$

If this were a deterministic problem, we would formulate it as

$$\max_{(x_t,y_t),t=0,\ldots,T} \sum_{t=0}^{T} (p_t y_t - c_t x_t).$$

However, we often want to represent the demands $\hat{D}_{t+1}$ as being random at time t. We might even want to allow our prices p_t, and perhaps even our costs c_t, to vary over time with both predictable (e.g. seasonal) and stochastic (uncertain) patterns. In this case, we are going to need to define a state variable S_t that captures what we know at time t before we make our decisions x_t and y_t. Designing state variables is subtle, but for now we would assume that it would include R_t, p_t, c_t, as well as the demands D_t that have arisen during interval $(t-1, t)$.

Unlike the newsvendor problem, the inventory problem can be challenging even if the distribution of demand D_t is known. However, if it is unknown, then we may need to maintain a belief state B_t about the distribution of demand, or perhaps the expected profits when we place an order x_t.

The features of this problem allow us to create a family of problems:

Static data If the prices p_t and costs c_t are constant (which is to say that $p_t = p$ and $c_t = c$), with a known distribution of demand, then we have a stochastic optimization problem where the state is just $S_t = R_t$.

Dynamic data Assume the price p_t evolves randomly over time, where $p_{t+1} = p_t + \varepsilon_{t+1}$, then our state variable is $S_t = (R_t, p_t)$.

History-dependent processes Imagine now that our price process evolves according to

$$p_{t+1} = \theta_0 p_t + \theta_1 p_{t-1} + \theta_2 p_{t-2} + \varepsilon_{t+1},$$

then we would write the sate as $S_t = (R_t, (p_t, p_{t-1}, p_{t-2})$.

Learning process Now assume that we do not know the distribution of the demand. We might put in place a process to try to learn it, either from observations of demands or sales. Let B_t capture our belief about the distribution of demand, which may itself be a probability distribution. In this case, our state variable would be $S_t = (R_t, p_t, B_t)$.

Now, we are going to introduce the device of a *policy*, which is a rule for making decisions. Let $X^\pi(S_t)$ be our rule for how much we are going to order at time t, and let $Y^\pi(S_t)$ be our rule for how much we are going to sell. For example, we may be in a period where prices are low, and we prefer to hold back on sales so that we can use our inventory at a later period when we think prices are high. Designing these policies is a major theme of this book, but for now we can write our optimization problem as

$$\max_\pi \mathbb{E} \sum_{t=0}^{T} (p_t Y^\pi(S_t) - c_t X^\pi(S_t)).$$

Here, our search over policies π means searching over both buying policies $X^\pi(S)$ and selling policies $Y^\pi(S)$.

2.3.2.2 Inventory planning with forecasts An important extension that arises in many real applications is where the data (demands, prices, even costs) may follow time-varying patterns which can be approximately forecasted. Let

$f_{tt'}^W$ = Forecast of some activity (demands, prices, costs) made at time t that we think will happen at time t'.

Forecasts evolve over time. They may be given to us from an exogenous source (a forecasting vendor), or we may use observed data to do our own updating of forecasts.

Assuming they are provided by an external vendor, we might describe the evolution of forecasts using

$$f^W_{t+1,t'} = f^W_{tt'} + \hat{f}^W_{t+1,t'},$$

where $\hat{f}^W_{t+1,t'}$ is the (random) change in the forecasts over all future time periods t'.

When we have forecasts, the vector $f^W_t = (f^W_{tt'})_{t' \geq t}$ technically becomes part of the state variable. When forecasts are available, the standard approach is to treat these as latent variables, which means that we do not explicitly model the evolution of the forecasts, but rather just treat the forecast as a static vector. We will return to this in chapter 9.

2.3.2.3 Lagged decisions There are many applications where we make a decision at time t (say, ordering new inventory) that does not arrive until time t' (as a result of shipping delays). In global logistics, these lags can extend for several months. For an airline ordering new aircraft, the lags can span several years.

We can represent lags using the notation

$x_{tt'}$ = Inventory ordered at time t to arrive at time t'.

$R_{tt'}$ = Inventory that has been ordered at some time before t that is going to arrive at time t'.

The variable $R_{tt'}$ is how we capture the effect of previous decisions. We can roll these variables up into the vectors $x_t = (x_{tt'})_{t' \geq t}$ and $R_t = (R_{tt'})_{t' \geq t}$.

Lagged problems are particularly difficult to model. Imagine that we want to sign contracts to purchase natural gas in month t'' that might be three years into the future to serve uncertain demands. This decision has to consider the possibility that we may place an order $x_{t't''}$ at a time t' that is between now (time t) and time t''. At time t, the decision $x_{t't''}$ is a random variable that depends not just on the price of natural gas at time t', but also the decisions we might make between t and t', as well as evolving forecasts.

2.3.3 Shortest path problems

Shortest path problems represent a particularly elegant and powerful problem class, since a node in the network can represent any discrete state, while links out of the node can represent a discrete action.

2.3.3.1 A deterministic shortest path problem A classical sequential decision problem is the shortest path problem. Let

$\mathcal{I}$ = The set of nodes (intersections) in the network,

$\mathcal{L}$ = The set of links (i,j) in the network,

c_{ij} = The cost (typically the time) to drive from node i to node j, $i,j \in \mathcal{I}, (i,j) \in \mathcal{L}$,

$\mathcal{I}^+_i$ = The set of nodes j for which there is a link $(i,j) \in \mathcal{L}$,

$\mathcal{I}^-_j$ = The set of nodes i for which there is a link $(i,j) \in \mathcal{L}$.

A traveler at node i needs to choose the link (i,j) where $j \in \mathcal{I}^+_i$ is a downstream node from node i. Assume that the traveler needs to get from an origin node q to a destination node r at least cost. Let

v_j = The minimum cost required to get from node j to node r.

Step 0. Let

$$v_j^0 \quad = \quad \begin{cases} \mathrm{M} & j \neq r, \\ 0 & j = r. \end{cases}$$

where "M" is known as "big-M" and represents a large number. Let $n = 1$.

Step 1. Solve for all $i \in \mathcal{I}$,

$$v_i^n = \min_{j \in \mathcal{I}_i^+} \left(c_{ij} + v_j^{n-1} \right).$$

Step 2. If $v_i^n < v_i^{n-1}$ for any i, let $n = n + 1$ and return to step 1. Else stop.

Figure 2.3 A basic shortest path algorithm.

We can think of v_j as the value of being in state j. At optimality, these values will satisfy

$$v_i = \min_{j \in \mathcal{I}_i^+} \big(c_{ij} + v_j \big).$$

This fundamental equation underlies all the shortest path algorithms used in our GPS devices, although these have been heavily engineered to achieve the rapid response we have become accustomed to. A basic shortest path algorithm is given in 2.3, although this represents just the skeleton of what a real algorithm would look like.

Shortest path problems represent particularly attractive instances of dynamic programming problems because the state space is just the set of intersections describing a network. Even though there may be millions of intersections describing large networks such as those in the U.S., Europe and other major regions, this is still quite manageable (production algorithms use different heuristics to restrict the search). Finding the best path across a complex transportation network will seem like a toy after we consider some of the multidimensional problems that arise in other applications.

2.3.3.2 A stochastic shortest path problem We are often interested in shortest path problems where there is uncertainty in the cost of traversing a link. For our transportation example, it is natural to view the travel time on a link as random, reflecting the variability in traffic conditions on each link.

There are situations where the costs or times are random. To handle this new dimension correctly, we have to specify whether we see the outcome of the random cost on a link before or after we make the decision whether to traverse the link. If the actual cost is only realized after we traverse the link, then are decision at node x_i that we made when we are at node i would be written

$$x_i = \arg\min_{j \in \mathcal{I}_i^+} \mathbb{E} \left(\hat{c}_{ij} + v_j \right),$$

where the expectation is over the (assumed known) distribution of the random cost $\hat{c}_{ij}$. For this problem, our state variable S is simply the node at which we are located.

If we get to make our decision after we learn $\hat{c}_{ij}$, then our decision would be written

$$x_i = \mathbb{E} \left\{ \arg\min_{j \in \mathcal{I}_i^+} \left(\hat{c}_{ij} + v_j \right) \right\},$$

In this setting, the state variable S is given by $S = (i, (\hat{c}_{ij})_j)$ includes both our current node, but also the costs on links emanating from node i.

2.3.3.3 The nomadic trucker A special kind of shortest path problem arises in an industry called truckload trucking, where a single truck driver will pick up a load at A, drive it from A to B, drop it off at B, and then has to look for a new load (there are places to call in to get a list of available loads). The driver has to think about how much money he will make moving the load, but he then also has to recognize that the load will move him to a new city.

The driver is characterized at each point in time by his current or future location ℓ_t (which is a region of the country), his equipment type E_t which is the type of trailer he is pulling (which can change depending on the needs of the freight), his estimated time of arrival at ℓ_t (denoted by τ^{eta}), and the time τ_t that he has been away from his home (which is fixed, so we exclude it from the state variable). We roll these attributes into a vector r_t given by

$$r_t = (\ell_t, E_t, \tau^{eta}, \tau_t^{home}).$$

When the driver arrives at the destination of a load, he calls a freight broker and gets a set of loads $\mathcal{L}_t$ that he can choose from. This means that his state variable (the information just before he makes a decision), is given by

$$S_t = (r_t, \mathcal{L}_t).$$

The driver has to choose among a set of actions $\mathcal{A}_t = (\mathcal{L}_t, \text{“hold”})$ that includes the loads in the set $\mathcal{L}_t$, or doing nothing. Once the driver makes this choice, the set $\mathcal{L}_t$ is no longer relevant. His state immediately after he makes his decision is called the *post-decision state* $S_t^a = r_t^a$, which is updated to reflect the destination of the load, and the time he is expected to arrive at this location.

The natural way for a driver to choose which action to take is to balance the contribution of the action, which we write as $C(S_t, a_t)$, and the value of the driver in his “post-decision” state r_t^a. We might write this policy, which we call $A^\pi(S_t)$, using

$$A^\pi(S_t) = \arg\max_{a \in \mathcal{A}_t} \big(C(S_t, a) + \overline{V}_t^a(r_t^a) \big).$$

The algorithmic challenge is creating the estimates $\overline{V}_t^a(r_t^a)$, which is an example of what we will call a *value function approximation*. If the number of possible values of the driver attribute vector r_t^a was not too large, we could solve this problem using the same way we would solve the stochastic shortest path problem introduced in section 2.3.3.2. The hidden assumption in this problem is that the number of nodes is not too large (even a million nodes is considered manageable). When a “node” is a multidimensional vector r_t, then we may have trouble manipulating all the possible values this may take (another instances of the curse of dimensionality).

2.3.4 Pricing

Imagine that we are trying to determine the price of a product, and that we feel that we can model the demand for the product using a logistics curve given by

$$D(p|\theta) = \theta_0 \frac{e^{\theta_1 - \theta_2 p}}{1 + e^{\theta_1 - \theta_2 p}}.$$

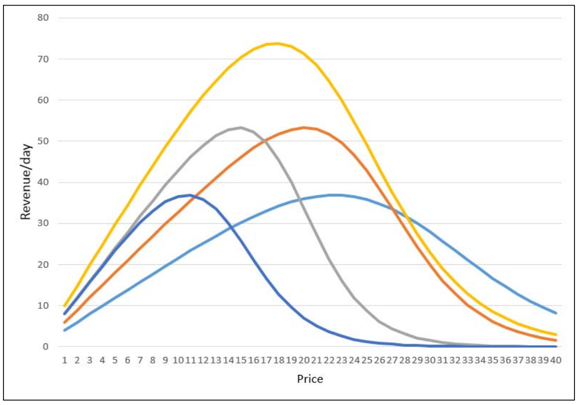


Figure 2.4 Illustration of a family of possible revenue curves.

The total revenue from charging price p is given by

$$R(p|\theta) = pD(p|\theta).$$

If we knew θ, finding the optimal price would be a fairly simple exercise. But now assume that we do not know θ. Figure 2.4 illustrates a family of potential curves that might describe revenue as a function of price.

We can approach this problem as one of learning the true value of θ. Let $\Theta = (\theta_1, \ldots, \theta_K)$ be a family of possible values of θ where we assume that one of the elements of Θ is the true value. Let p_k^n be the probability that $\theta = \theta_k$ after we have made n observations. The state of our learning system, then, is $S^n = (p_k^n)_{k=1}^K$ which captures our belief about θ.

2.3.5 Medical decision making

Physicians have to make decisions about patients who arrive with some sort of complaint. The process starts by taking a medical history which consists of a series of questions about the patients history and lifestyle. Let a^n be this history captured as a set of attributes, where a^n might consist of thousands of different possible characteristics (humans are complicated!). The physician might then order additional tests which produce additional information, or she might prescribe medication or request a surgical procedure. Let d^n capture these decisions. We can wrap this combination of patient attributes a^n and medical decisions d^n into a set of explanatory variables that we designate $x^n = (a^n, d^n)$.

Now assume we observe an outcome y^n which for simplicity we are going to represent as binary, where $y^n = 1$ can be interpreted as "success" and $y^n = 0$ is a "failure." We are going to assume that we can model the random variable y^n (random, that is, before we observe the results of the treatment) using a logistic regression, which is given by

$$\mathbb{P}[y^n = 1 | x^n = (a^n, d^n), \theta] = \frac{e^{\theta x^n}}{1 + e^{\theta x^n}}. \tag{2.37}$$

This problem illustrates two types of uncertainty. The first is the attributes of the patient a^n, where we typically would not have a probability distribution describing these attributes. It is difficult (actually, impossible) to develop a probabilistic model of the complex characteristics captured in a^n describing a person, since these attributes are going to exhibit complex correlations. By contrast, the random variable y^n has a well defined mathematical model, characterized by an unknown (and high dimensional) parameter vector θ.

Later in the volume, we are going to use two different approaches for handling these different types of uncertainty. For patient attributes, we are going to use an approach that is often known as *data driven*. We might have access to a large dataset of prior attributes, decisions and outcomes, that we might represent as $(a^n, d^n, y^n)_{n=1}^N$. Alternatively, we may assume that we simply observe a patient a^n (this is the data-driven part), then make a decision $d^n = D^\pi(S^n)$ using a decision function $D^\pi(S^n)$ that can depend on a state variable S^n, and then observe an outcome y^n which we can describe using our probability model.

2.3.6 Scientific exploration

Scientists, looking to discover new drugs, new materials, or new designs for a wing or rocket engine, are often faced with the need to run difficult laboratory experiments looking for the inputs and processes to produce the best results. Inputs might be a choice of catalyst, the shape of a nanoparticle, or the choice of molecular compound. There might be different steps in a manufacturing process, or the choice of a machine for polishing a lens.

Then, there are the continuous decisions. Temperatures, pressures, concentrations, ratios, locations, diameters, lengths and times are all examples of continuous parameters. In some settings these are naturally discretized, although this can be problematic if there are three or more continuous parameters we are trying to tune at the same time.

We can represent a discrete decision as choosing an element $x \in \mathcal{X} = \{x_1, \ldots, x_M\}$. Alternatively, we may have a continuous vector $x = (x_1, x_2, \ldots, x_K)$. Let x^n be our choice of x (whether it is discrete or continuous). We are going to assume that x^n is the choice we make *after* running the nth experiment that guides the $n+1st$ experiment, from which we observe W^{n+1}. The outcome W^{n+1} might be the strength of a material, the reflexivity of a surface, or the number of cancer cells killed.

We use the results of an experiment to update a belief model. If x is discrete, imagine we have an estimate $\bar{\mu}_x^n$ which is our estimate of the performance of running an experiment with choice x. If we choose $x = x^n$ and observe W^{n+1}, then we can use statistical methods (which we describe in chapter 3) to obtain updated estimates $\bar{\mu}_x^{n+1}$. In fact, we can use a property known as *correlated beliefs* that may allow us to run experiment $x = x^n$ and update estimates $\bar{\mu}_{x'}^{n+1}$ for values x' other than x.

Often, we are going to use some parametric model to predict a response. For example, we might create a linear model which can be written

$$f(x^n|\theta) = \theta_0 + \theta_1\phi_1(x^n) + \theta_2\phi_2(x^n) + \ldots, \tag{2.38}$$

where $\phi_f(x^n)$ is a function that pulls out relevant pieces of information from the inputs x^n of an experiment. For example, if element x_i is the temperature, we might have $\phi_1(x^n) = x_i^n$ and $\phi_2(x^n) = (x_i^n)^2$. If x_{i+1} is the pressure, we could also have $\phi_3(x^n) = x_i^n x_{i+1}^n$ and $\phi_4(x^n) = x_i^n (x_{i+1}^n)^2$.

Equation (2.38) is known as a linear model because it is linear in the parameter vector θ. The logistic regression model in (2.37) is an example of a nonlinear model (since it is nonlinear in θ). Whether it is linear or nonlinear, parametric belief models capture the structure of a problem, reducing the uncertainty from an unknown $\bar{\mu}_x$ for each x (where the number of different values of x can number in the thousands to millions or more) down to a set of parameters θ that might number in the tens to hundreds.

2.3.7 Statistics and machine learning

There are close parallels between stochastic optimization and machine learning. Let:

x^n = The data corresponding to the nth instances of a problem (the characteristics of a patient, the attributes of a document, the data for an image) that we want to use to predict an outcome y^n,

y^n = The response, which might be the response of a patient to a treatment, the categorization of a document, or the classification of an image,

$f(x^n|\theta)$ = Our model which we use to predict y^n given x^n,

θ = An unknown parameter vector used to determine the model.

We assume we have some metric that indicates how well our model $f(x|\theta)$ is performing. For example, we might use

$$L(x^n, y^n|\theta) \quad = \quad (y^n - f(x^n|\theta))^2.$$

The function $f(x|\theta)$ can take on many forms. The simplest is a basic linear model of the form

$$f(x|\theta) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(x),$$

where $\phi_f(x)$ is known as a feature, and $\mathcal{F}$ is the set of features. There may be just a handful of features, or thousands. The statistics and machine learning communities have developed a broad array of functions, each of which is parameterized by some vector θ (often designated as weights w). We review these in some depth in chapter 3.

The most typical optimization problem in machine learning is to use a batch dataset to solve the nonlinear problem

$$\min_{\theta} \frac{1}{N} \sum_{n=1}^{N} (y^n - f(x^n|\theta))^2.$$

For problems where the dimensionality of θ is in the thousands, this can be a difficult algorithmic challenge.

There is a separate class of machine learning problems that involve working with data that arrives over time, a field that is known as *online learning*. For this setting, we index our sequence of data and observations as $(x^0, y^1), (x^1, y^2), \ldots, (x^n, y^{n+1})$. This indexing communicates the property that we see x^n before observing y^{n+1}. In this case, we would let θ^n be our estimate of θ based on the data $(x^0, y^1, \ldots, x^n)$. There are a variety of algorithms that we will later describe as policies $\Theta^{\pi}(S^n)$ which maps what we know as of time n (in theory this could be the entire history, but typically it is more compact) to produce a parameter estimate θ^n to be used to predict y^{n+1}. Now we have the problem of designing policies, which is the focus of most of this book.

2.4 BIBLIOGRAPHIC NOTES

- Section xx -

PROBLEMS

2.1 Recall our newsvendor problem

$$\max_x \mathbb{E}_W F(x,W)$$

where $F(x,W) = p\min(x,W) - cx$. Assume that W is given by a known distribution $f^W(w)$ with cumulative distribution

$$F^W(w) = \mathbb{P}[W \leq w].$$

You are going to show that the optimal solution x^* satisfies

$$F^W(x^*) = \frac{p-c}{p}. \tag{2.39}$$

Do this by first finding the stochastic gradient $\nabla_x F(x,W)$ which will give you a gradient that depends on whether $W < x$ or $W > x$. Now take the expected value of this gradient and set it equal to zero, and use this to show (2.39).

2.2 Newsvendor with random prices. Discuss modeling and convexity. Create version with sales as a decision variable and show that the resulting problem is convex.

CHAPTER 3

LEARNING IN STOCHASTIC OPTIMIZATION

3.1 BACKGROUND

It is useful to begin our discussion of statistical learning by describing the learning issues that arise in the context of stochastic optimization. This section provides an overview of the following dimensions of learning problems:

- Observations and data in stochastic optimization - While classical statistical learning problems consists of datasets comprised of input (or independent) variables x^n and output (or dependent) variables y^n, in stochastic optimization the input variables x^n are decisions that we control.

- Functions we are learning - There are a half dozen different classes of functions that we may need to approximate in different stochastic optimization contexts.

- Approximation strategies - Here we summarize the three major classes of approximation strategies from the statistical learning literature. The rest of this chapter summarizes these strategies.

- Objectives - Sometimes we are trying to fit a function to data which minimizes errors, and sometimes we are finding a function to maximize contributions or minimize costs. Either way, learning functions is always its own optimization problem, sometimes buried within a larger stochastic optimization problem.

- Batch vs. recursive learning - Most of the statistical learning literature focuses on using a given dataset (and of late, these are very large datasets) to fit complex statistical models. In stochastic optimization, we primarily depend on adaptive learning, so this chapter describes recursive learning algorithms.

3.1.1 Observations and data in stochastic optimization

Before we present our overview of statistical techniques, we need to say a word about the data we are using to estimate functions. In statistical learning, it is typically assumed that we are given input data x, after which we observe a response y. Some examples include

- We may observe the characteristics x of a patient to predict the likelihood y of contracting cancer, or the result of a treatment regime.
- We wish to predict the weather y based on meteorological conditions x that we observe now.
- We observe the pricing behavior of nearby hotels, given by x, to predict the response y of whether a customer books a room, y, in our hotel at a given price.

In these settings, we obtain a dataset where we associate the response y^n with the observations x^n, which gives us a dataset $(x^n, y^n)_{n=1}^N$.

In stochastic optimization, x is a decision, such as a choice of drug treatment, the price of a product, the inventory of vaccines, or the choice of movies to display on a user's internet account. We view x as controllable, although it can consist of a mixture of controllable elements (such as a drug dosage), and uncontrollable elements (the characteristics of the patient). After choosing x^n, we then observe y^{n+1}. We change the indexing so that the index n (this could also be time t) indicates the information content. So, the decision x^n is not able to "see" the outcome y^{n+1}. This notation is more natural for sequential learning, as arises in stochastic optimization, than the batch datasets more familiar in statistics and machine learning. Further, we start at $n = 0$, where x^0 is the first decision, which we have to make before seeing any observations.

Finally, much of our "data" is going to come from a class of methods known as Monte Carlo simulation. This is a computational technique used for sampling from a distribution. We introduce this field in considerably more depth in chapter 10. In a nutshell, imagine that we have a mathematical function (this could be an analytical function, or a complex computer simulation), which depends on controllable inputs x, and a series of random (uncontrollable) inputs W. For example, W might represent a sequence of requests by customers who log into the website of a hotel looking to book a room. Now let y be the number who actually book a room, which reflects the price that the hotel sets (which is the decision x). We can represent the sequence of decisions x, information W and observations y as

$$(x^0, W^1, y^1, x^1, W^2, y^2, x^2, \ldots, x^n, W^{n+1}, y^{n+1}, \ldots, W^N, y^N).$$

Often, it will be more natural to index using time. In this case, we would write

$$(x_0, W_1, y_1, x_1, W_2, y_2, x_2, \ldots, x_t, W_{t+1}, y_{t+1}, \ldots, W^T, y^T).$$

In stochastic optimization, we are going to design various ways of determining what x should be. Sometimes, we will do this using traditional observations of a physical process (the best examples of this are found in chapter 7 on derivative-free stochastic optimization). Often, we are doing this in a computer, where we have to simulate W rather than observe it (or observe y directly). We can create samples of W using Monte Carlo simulation, which is easily the most powerful computational tool for modeling stochastic systems.

3.1.2 Functions we are learning

The need to approximate functions arise in a number of settings in stochastic optimization. Some of the most important include:

1) Approximating the expectation of a function $\mathbb{E}F(x, W)$ to be maximized, where we assume that we have access to unbiased observations $\hat{F} = F(x, W)$ for a given decision x, which draws on a major branch of statistical learning known as supervised learning.

2) Creating an approximate policy $X^\pi(S|\theta)$ (or $A^\pi(S|\theta)$ or $U^\pi(S|\theta)$). We may fit these functions using one of two ways. We may assume that we have an exogenous source of decisions x that we can use to fit our policy $X^\pi(S|\theta)$ (this would be supervised learning). More frequently, we are tuning the policy to maximize a contribution (or minimize a cost), which is sometimes referred to as reinforcement learning.

3) Approximating the value of being in a state S, which we first saw in section 2.1.9. Let $V_t(S_t)$ be the value of being in state S_t at time t and then following some policy from time t onward. We wish to find an approximation $\overline{V}_t(S_t)$ that approximates $V_t(S_t)$. This is supervised learning, but requires that we estimate our value function using biased observations (we get into this in much more detail in chapters 17 and 18).

4) Later we will introduce a class of policies that we call *parametric cost function approximations* where we have to learn two types of functions:

4a) Parametric modifications of cost functions (for example, a penalty for not serving a demand now but instead holding it for the future).

4b) Parametric modifications of constraints (for example, inserting schedule slack into an airline schedule to handle uncertainty in travel times).

Each of these parametric modifications have to be tuned (which is a form of function estimation) to produce the best results over time.

5) Learning any of the underlying models in a dynamic system. These include:

5a) The *transition function* that describes how the system evolves over time, which we will write as $S^M(S_t, x_t, W_{t+1})$ which is used to compute the next state S_{t+1}. This arises in complex environments where the dynamics are not known, such as modeling how much water is retained in a reservoir, which depends in a complex way on rainfall and temperature. We might approximate losses using a parametric model that has to be estimated.

5b) The cost or contribution functions (also known as rewards, gains, losses, utility function). This might be unknown if a human is making a decision to maximize an unknown utility, which we might represent as a linear model with parameters to be determined from observed behaviors.

5c) The evolution of exogenous quantities such as wind or prices, where we might model an observation W_{t+1} as a function of the history $W_t, W_{t-1}, W_{t-2}, \ldots$, where we have to fit our model from past observations.

The first case is the simplest and purest setting, which involves estimating a function $\overline{F}$

$$\overline{F}(x) \approx \mathbb{E}F(x, W),$$

based on noisy observations of the function $\hat{F}^n$ which we assume are unbiased observations of $\mathbb{E}F(x, W)$. In this problem, we have to learn the function $\overline{F}(x)$ so that we can solve the optimization problem

$$\max_x \mathbb{E}F(x, W). \tag{3.1}$$

The goal is to find an accurate approximation $\overline{F}$ so that we can replace the optimization problem (3.1) with

$$\max_x \overline{F}(x).$$

This approach is typically performed in the context of an iterative algorithm where we solve an approximation $\overline{F}^n(x)$ obtained after n iterations, and use this to guide the choice of what to observe in the $n + 1st$ iteration. Having access to the unbiased but sampled estimates $\hat{F}^n$ of $\mathbb{E}F(x, W)$ means that this is a form of *supervised learning*.

The second setting is where we wish to find a policy $X^\pi(S_t)$ that gives us an action $x_t = X^\pi(S_t)$. Later we are going to show that there are four major classes of policies, one of which, *policy function approximations*, come in the form of parametric (or sometimes non-parametric) functions. For example, assume that the state consists of $S_t = (S_{t1}, S_{t2}, S_{t3})$ where S_{t1} might be, for example, the amount of oil in storage, S_{t2} might be the forecast of temperatures (indicating whether the upcoming winter will be unusually cold), and S_{t3} might be a global economic variable. We may then be trying to estimate how many barrels of oil we should store. We may try to write our decision function as

$$X^\pi(S_t|\theta) = \theta_0 + \theta_1 S_{t1} + \theta_2 S_{t2} + \theta_3 S_{t3} + \theta_4 S_{t1}^2 + \theta_5 S_{t1} S_{t3}.$$

We then tune θ by solving the problem

$$\max_\theta \mathbb{E} \sum_{t=0}^{T} C(S_t, X^\pi(S_t|\theta)). \tag{3.2}$$

This is a classical stochastic optimization problem which we first cover in chapters 4, 5 (for derivative-based stochastic search), and 7 (for derivative-free stochastic search). This would be unsupervised learning, but is sometimes referred to as *reinforcement learning* (a term that originated in the setting of approximating value functions for dynamic programs).

Equation (3.2) is the most widely used starting point for finding policies, but there are settings where we are simply trying to mimic another decision-maker (possibly human, but perhaps a computationally complex model). If we had access to a source of decisions, that source would be known as the “supervisor” (which is the origin of the term). However, we aspire to doing even better.

The third and most difficult setting is when we need to estimate the value $V_t(S_t)$ of starting in a state S_t, typically following some rule or policy from time t onward. If we can write the state S_{t+1} as a function of the current state S_t, a decision x_t, and some new random information W_{t+1}, then we can find a good (possibly optimal) action x_t by solving

$$\max_{x_t} \left(C(S_t, x_t) + \mathbb{E}\{\overline{V}_{t+1}(S_{t+1})|S_t\} \right), \tag{3.3}$$

where $\overline{V}_{t+1}(S_{t+1})$ is a statistical approximation of $V_{t+1}(S_{t+1})$. A challenge we almost always incur in this setting is that we have to approximate $\overline{V}_{t+1}(S_{t+1})$ using biased estimates of the value of being in state S_{t+1}. This is the basis of an algorithmic strategy known broadly as *approximate dynamic programming* (or reinforcement learning) which we introduce starting in chapters 17 and 18.

The unknown function $\overline{F}(x)$, the value function $\overline{V}_t(S_t)$, and the decision function $X^\pi(S_t|\theta)$, are all examples of functions that we may need to estimate using statistical methods. This chapter is an overview of some of the tools from statistics/machine learning that can be used, where we have limited our attention to methods that can be implemented in a recursive manner. This is an important problem setting, since it complicates the use of popular methods such as Lasso, an important statistical method that helps to identify the most important variables in models where there is a large library of potential explanatory variables.

3.1.3 Approximation strategies

Our tour of statistical learning makes a progression through the following methods:

Lookup tables Here we estimate a function $f(x)$ where x falls in a discrete region $\mathcal{X}$ given by a set of points $x_1, x_2, \ldots, x_M$. A point x_m could be the characteristics of a person, a type of material, or a movie. Or it could be a point in a discretized, continuous region. In most applications, lookup tables work well in one or two dimensions, then become difficult (but feasible) in three or four dimensions, and then quickly become impractical starting at four or five dimensions (this is the classical "curse of dimensionality"). Our presentation focuses on using aggregation, and especially hierarchical aggregation, both to handle the curse of dimensionality, but also to manage the transition in recursive estimation from initial estimates with very little data, to produce better estimates as more data becomes available.

Parametric models There are many problems where we can approximate a function using an analytical model in terms of some unknown parameters. The simplest is a linear model that might look like

$$f(x|\theta) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots. \tag{3.4}$$

Often it is useful to define a set of *features* $\phi_f(x),\ f \in \mathcal{F}$ where $\phi_f(x)$ extracts a specific piece of information from x which could be a vector, or the data describing a movie or ad. Equation (3.4) is called a linear model because it is linear in θ (it may be highly nonlinear in x). Alternatively, we may have a nonlinear model such as

$$f(x|\theta) = e^{\sum_{f \in \mathcal{F}} \theta_f \phi_f(x)}.$$

Parametric models may be low-dimensional (1-100 parameters), or high-dimensional (e.g. several hundred to thousands of parameters).

Nonparametric models Nonparametric models create estimates by building a structure directly from the data. A simple example is where we estimate $f(x)$ from a weighted combination of nearby observations drawn from a set $(f^n, x^n),\ n = 1, \ldots, N$.

Notably missing from this chapter is approximation methods for convex functions. There are many applications where $F(x,W)$ is convex in x. This function is so special that we defer handling this problem class until chapter 5 (and especially chapter 19) when we address stochastic convex (or concave) stochastic optimization problems such as linear programs with random data.

We begin our presentation with lookup tables, which are the simplest way to represent a function without assuming any structure. We begin by presenting lookup tables from both frequentist and Bayesian perspectives. Bayesian and frequentist approaches have two fundamental differences. First, Bayesian statistics starts with a prior distribution of belief about a parameter (or set of parameters). This is useful in settings where obtaining an observation of the function is expensive (it might involve an expensive simulation, or a laboratory or field experiment). We may even need to choose what observation to make before we have collected any data.

Second, a Bayesian belief about a set of parameters is always in the form of a probability distribution. In Bayesian statistics, the truth is viewed as a random variable, where experiments refine the distribution, converging (with enough experiments) to a single truth. By contrast, in the frequentist perspective, there is one truth, and we use the variability in the observations to make statistical statements about where the truth lies.

In stochastic optimization, we need both belief models. As a general rule, Bayesian models are best when we have access to some prior, and where function evaluations are expensive (remember, that a "function" is not always a mathematical function - it might be a field experiment). By contrast, frequentist models tend to be useful when we have access to a relatively large number of observations (typically where data is not that difficult to collect), and where we do not have access to a prior.

3.1.4 Objectives

Learning in stochastic optimization comes in two forms:

- Learning a function - We might want to learn an approximation of a function such as an objective function $\mathbb{E}F(x,W)$, or a value function $V(s)$ or perhaps even the transition function $S^M(s,x,W)$. In these settings, we assume we have a source of observations of our function that may be noisy, and even biased. For example, we might have access to $\hat{f}^{n+1}$ which is a noisy observation of $\mathbb{E}F(x^n,W)$ that we are going to approximate with some function $f(x|\theta)$. If we collect a dataset $(x^0,\hat{f}^1,x^1,\hat{f}^2,\ldots,x^{n-1},\hat{f}^n)$, we would look to find θ that minimizes the error between the observations $\hat{f}$ and $f(x|\theta)$ using

$$\min_\theta \frac{1}{n}\sum_{m=0}^{n-1}(\hat{f}^m - f(x^{m+1}|\theta)^2. \tag{3.5}$$

- Maximizing rewards (or minimizing costs) - We can search for a policy $X^\pi(s|\theta)$ that maximizes a contribution function $C(s,x)$ using

$$\max_\theta \sum_{n=0}^{N} C(s^n, X^\pi(s^n|\theta)), \tag{3.6}$$

where the states evolve according to a known transition function $s^{n+1}=S^M(s^n,x^n,W^{n+1})$.

3.1.5 Batch vs. recursive learning

Equation (3.5) (or (3.6)) are the standard problems that arise in batch learning problems, where we use a fixed dataset (possibly a very large one in the modern era of "big data") to fit a model (increasingly high-dimensional models such as neural networks which we introduce below).

While batch learning can arise in stochastic optimization, the most common learning problems are adaptive. Imagine that after n iterations (or samples), we have the sequence

$$(x^0, W^1, y^1, x^1, W^2, y^2, x^2, \dots, W^n, y^n).$$

Assume that we use this data to obtain an estimate of our function that we call $\overline{F}^n(x)$. Now assume we use this estimate to make a decision x^n, after which we experience exogenous information W^{n+1} and then the response y^{n+1}. We need to use our prior estimate $\overline{F}^n(x)$ along with the new information (W^{n+1}, y^{n+1}) to produce a new estimate $\overline{F}^{n+1}(x)$.

We could, of course, just solve a new batch problem with one more observation, most of the time we are going to be able to do this recursively, where our history is captured in a compact way. In fact, this chapter focuses primarily on recursive learning.

3.2 ADAPTIVE LEARNING USING EXPONENTIAL SMOOTHING

The most common method we will use for adaptive learning is known by various names, but is popularly referred to as *exponential smoothing*. Assume we have a sequence of observations of some quantity, which might be the number of people booking a room, the response of a patient to a particular drug, or the travel time on a path. Let μ be the unknown truth, which could be the average number of people booking a room at a particular price, or the probability a patient responds to a drug, or the average travel time of our path. We want to estimate the average from a sequence of observations.

Let W^n be the nth observation of the quantity we are trying to estimate, and let $\bar{\mu}^n$ be our estimate of the true mean μ after n observations. The most widely used method for computing $\bar{\mu}^{n+1}$ given $\bar{\mu}^n$ and a new observation W^{n+1} is given by

$$\bar{\mu}^{n+1} = (1 - \alpha_n)\bar{\mu}^n + \alpha_n W^{n+1}. \tag{3.7}$$

The variable α_n is known variously as a learning rate, smoothing factor or (in this book) a stepsize (we will see the motivation for the term stepsize in chapter 5).

Equation (3.7) is so intuitive that people often use it without realizing that it has a foundation in formal optimization concepts. Assume that $f(x)$ is some nonlinear function in x (which may be a vector), with gradient $\nabla_x f(x)$ given by

$$\nabla_x f(x) = \begin{matrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_K} \end{matrix} .$$

If we wish to minimize $f(x)$ (which we will assume is a nice, smooth, concave function) we might use a classical steepest ascent algorithm with a basic update

$$x^{n+1} = x^n - \alpha_n \nabla_x f(x^n). \tag{3.8}$$

The vector $\nabla_x f(x)$ points in the direction of steepest ascent, so we use $-\nabla_x f(x)$ to point in the direction of steepest descent. Now we need to determine how far to move in this direction. This is determined by α_n, which is the reason that it is often called a stepsize. If we know the function $f(x)$, we would find the stepsize by solving

$$\alpha_n = \arg\max_{\alpha \geq 0} f(x^n - \alpha \nabla_x f(x^n)). \tag{3.9}$$

We can use this idea to formulate our problem of finding the best estimate $\bar{\mu}$ of the mean of our random variable W as an optimization problem using

$$\min_\mu f(\mu) = \mathbb{E}\frac{1}{2}(\mu - W)^2. \tag{3.10}$$

If we could compute the expectation, we could find $f(\mu)$ and use our steepest descent algorithm. Since we cannot compute $f(\mu)$, we are going to use a simple adaptation known as a stochastic gradient, where we pick

Let $f(\mu) = \mathbb{E}\frac{1}{2}(\mu - W)^2$, and let $f(\mu, W) = \frac{1}{2}(\mu - W)^2$. We cannot compute the expectation needed to find the function $f(\mu)$, which prevents us from directly applying our steepest ascent algorithm. However, we can still use the basic idea, where we replace the gradient with a "stochastic gradient." Assume our current estimate of μ is $\bar{\mu}^n$, and then we run the $n + 1st$ experiment and obtain W^{n+1}. Now compute $\nabla f(\bar{\mu}^n, W^{n+1})$ which, for our problem, is given by

$$\nabla_\mu f(\mu, W)\big|_{\mu=\bar{\mu}^n, W=W^{n+1}} = (\bar{\mu}^n - W^{n+1}). \tag{3.11}$$

Equation (3.11) is called a "stochastic gradient" because it depends on the random variable W^{n+1}. This is a bit misleading, because we only compute (3.11) after W^{n+1} becomes known. We still use it in the same way as our steepest descent algorithm in equation (3.9), allowing us to write

$$\begin{aligned} \bar{\mu}^{n+1} &= \bar{\mu}^n + \alpha_n \nabla F(\bar{\mu}^n, W^{n+1} && (3.12)\\ &= \bar{\mu}^n + \alpha_n(\bar{\mu}^n - W^{n+1}) && (3.13)\\ &= (1 - \alpha_n)\bar{\mu}^n + \alpha_n W^{n+1}. && (3.14) \end{aligned}$$

This brief derivation shows us that our simple exponential smoothing method in equation (3.7) is really the same as our stochastic gradient algorithm.

This just leaves the question: how do we find the stepsize? So, it turns out that we can no longer find the optimal stepsize by solving the optimization problem in (3.9), primarily because we cannot compute the expectation $\mathbb{E}f(\mu, W)$, which means we cannot find the function $f(\mu)$. The good news is that we are limited to relatively simple formulas. For example, we will often be using

$$\alpha_n = \frac{1}{n}.$$

In other settings, we may just use $\alpha_n = \alpha$ where $0 < \alpha \leq 1$ is a fixed parameter. Larger values of α naturally put more weight on more recent observations, which is a desirable property if the data W^n is coming from a physical system that is changing dynamically over time. For example, imagine that W_n is the price of a stock responding to market forces. We could use equation (3.7) to produce a smoothed estimate, with larger values of α putting more weight on recent data so that it responds to recent conditions (but at a price of introducing more volatility into the estimate $\bar{\mu}^n$). Chapter 6 is an entire chapter dedicated to different stepsize formulas.

Equation (3.7) is often referred to as exponential smoothing, although it arises in so many settings that it has picked up a number of names. It is easily the most widely used formula in stochastic optimization and learning.

3.3 LOOKUP TABLES WITH FREQUENTIST UPDATING

The frequentist view is arguably the approach that is most familiar to people with an introductory course in statistics. Assume we are trying to estimate the mean μ of a random variable W which might be the performance of a device or policy. Let W^n be the nth sample observation, such as the sales of a product or the blood sugar reduction achieved by a particular medication. Also let $\bar{\mu}^n$ be our estimate of μ, and $\hat{\sigma}^{2,n}$ be our estimate of the variance of W. We know from elementary statistics that we can write $\bar{\mu}^n$ and $\hat{\sigma}^{2,n}$ using

$$\bar{\mu}^n = \frac{1}{n}\sum_{m=1}^{n} W^m \tag{3.15}$$

$$\hat{\sigma}^{2,n} = \frac{1}{n-1}\sum_{m=1}^{n}(W^m - \bar{\mu}^n)^2. \tag{3.16}$$

The estimate $\bar{\mu}^n$ is a random variable (in the frequentist view) because it is computed from other random variables, namely $W^1, W^2, \ldots, W^n$. Imagine if we had 100 people each choose a sample of n observations of W. We would obtain 100 different estimates of $\bar{\mu}^n$, reflecting the variation in our observations of W. The best estimate of the variance of the estimator $\bar{\mu}^n$ is easily found to be

$$\bar{\sigma}^{2,n} = \frac{1}{n}\hat{\sigma}^{2,n}.$$

Note that as $n \rightarrow \infty$, $\bar{\sigma}^{2,n} \rightarrow 0$, but $\hat{\sigma}^{2,n} \rightarrow \sigma^2$ where σ^2 is the true variance of W. If σ^2 is known, there would be no need to compute $\hat{\sigma}^{2,n}$ and $\bar{\sigma}^{2,n}$ would be given as above with $\hat{\sigma}^{2,n} = \sigma^2$.

We can write these expressions recursively using

$$\bar{\mu}^n = \left(1 - \frac{1}{n}\right)\bar{\mu}^{n-1} + \frac{1}{n}W^n, \tag{3.17}$$

$$\hat{\sigma}^{2,n} = \begin{cases} \frac{1}{n}(W^n - \bar{\mu}^{n-1})^2 & n = 2, \\ \frac{n-2}{n-1}\hat{\sigma}^{2,n-1} + \frac{1}{n}(W^n - \bar{\mu}^{n-1})^2 & n > 2. \end{cases} \tag{3.18}$$

We will often speak of our belief state which captures what we know about the parameters we are trying to estimate. Given our observations, we would write our belief state as

$$B^n = \left(\bar{\mu}^n, \hat{\sigma}^{2,n}, n\right).$$

Equations (3.17) and (3.18) describe how our belief state evolves over time.

3.4 LOOKUP TABLES WITH BAYESIAN UPDATING

The Bayesian perspective casts a different interpretation on the statistics we compute which is particularly useful in the context of learning when observations are expensive (imagine having to run expensive simulations or field experiments). In the frequentist perspective, we do not start with any knowledge about the system before we have collected any data. It is easy to verify from equations (3.17) and (3.18) that we never use $\bar{\mu}^0$ or $\hat{\sigma}^{2,0}$.

By contrast, in the Bayesian perspective we assume that we begin with a prior distribution of belief about the unknown parameter μ. In other words, any number whose value we do not know is interpreted as a random variable, and the distribution of this random variable represents our belief about how likely μ is to take on certain values. So if μ is the true but unknown mean of W, we might say that while we do not know what this mean is, we think it is normally distributed around θ^0 with standard deviation σ^0. Thus, the true mean μ is treated as a random variable with a known mean and variance, but we are willing to adjust our estimates of the mean and variance as we collect additional information. If we add a distributional assumption such as the normal distribution, we would say that this is our initial distribution of belief, known generally as the Bayesian prior.

The Bayesian perspective is well suited to problems where we are collecting information about a process where observations are expensive. This might arise when trying to price a book on the internet, or plan an expensive laboratory experiment. In both cases, we can be expected to have some prior information about the right price for a book, or the behavior of an experiment using our knowledge of physics and chemistry.

We note a subtle change in notation from the frequentist perspective, where $\bar{\mu}^n$ was our statistic giving our estimate of μ. In the Bayesian view, we let $\bar{\mu}^n$ be our estimate of the mean of the random variable μ after we have made n observations. It is important to remember that μ is a random variable whose distribution reflects our prior belief about μ. The parameter $\bar{\mu}^0$ is not a random variable. This is our initial estimate of the mean of our prior distribution. After n observations, $\bar{\mu}^n$ is our updated estimate of the mean of the random variable μ (the true mean).

Below we first use some simple expressions from probability to illustrate the effect of collecting information. We then give the Bayesian version of (3.17) and (3.18) for the case of independent beliefs, where observations of one choice do not influence our beliefs about other choices. We follow this discussion by giving the updating equations for correlated beliefs, where an observation of μ_x for alternative x tells us something about $\mu_{x'}$. We round out our presentation by touching on other important types of distributions.

3.4.1 The updating equations for independent beliefs

We begin by assuming (as we do through most of our presentation) that our random variable W is normally distributed. Let σ^2_W be the variance of W, which captures the noise in our ability to observe the true value. To simplify the algebra, we define the *precision* of W as

$$\beta^W = \frac{1}{\sigma^2_W}.$$

Precision has an intuitive meaning: smaller variance means that the observations will be closer to the unknown mean, that is, they will be more precise. Now let $\bar{\mu}^n$ be our estimate of the true mean μ after n observations, and let β^n be the precision of this estimate. That is, having already seen the values $W^1, W^2, ..., W^n$, we believe that the mean of μ is $\bar{\mu}^n$, and the variance of μ is $\frac{1}{\beta^n}$. We say that we are "at time n" when this happens; note that all quantities that become known at time n are indexed by the superscript n, so the observation W^{n+1} is not known until time $n+1$. Higher precision means that we allow for less variation in the unknown quantity; that is, we are more sure that μ is close to $\bar{\mu}^n$.

After observing W^{n+1}, the updated mean and precision of our estimate of μ is given by

$$\bar{\mu}^{n+1} = \frac{\beta^n \bar{\mu}^n + \beta^W W^{n+1}}{\beta^n + \beta^W}, \tag{3.19}$$

$$\beta^{n+1} = \beta^n + \beta^W. \tag{3.20}$$

Equation (3.19) can be written more compactly as

$$\bar{\mu}^{n+1} = (\beta^{n+1})^{-1} \left(\beta^n \bar{\mu}^n + \beta^W W^{n+1} \right). \tag{3.21}$$

There is another way of expressing the updating which provides insight into the structure of the flow of information. First define

$$\begin{aligned} \tilde{\sigma}^{2,n} &= \mathit{Var}^n[\bar{\mu}^{n+1}] \\ &= \mathit{Var}^n[\bar{\mu}^{n+1} - \bar{\mu}^n] \end{aligned} \tag{3.22}$$

where $\mathit{Var}^n[\cdot] = Var[\cdot \,|\, W^1, ..., W^n]$ denotes the variance of the argument given the information we have through n observations. For example,

$$\mathit{Var}^n[\bar{\mu}^n] = 0$$

since, given the information after n observations, $\bar{\mu}^n$ is a number that we can compute deterministically from the prior history of observations.

The parameter $\tilde{\sigma}^{2,n}$ can be described as the variance of $\bar{\mu}^{n+1}$ given the information we have collected through iteration n, which means the only random variable is W^{n+1}. Equivalently, $\tilde{\sigma}^{2,n}$ can be thought of as the *change* in the variance of $\bar{\mu}^n$ as a result of the observation of W^{n+1}. Equation (3.22) is an equivalent statement since, given the information collected up through iteration n, $\bar{\mu}^n$ is deterministic and is therefore a constant. We use equation (3.22) to offer the interpretation that $\tilde{\sigma}^{2,n}$ is the *change* in the variance of our estimate of the mean of μ.

It is possible to write $\tilde{\sigma}^{2,n}$ in different ways. For example, we can show that

$$\tilde{\sigma}^{2,n} = \sigma^{2,n} - \sigma^{2,n+1} \tag{3.23}$$

$$= \frac{(\sigma^{2,n})}{1 + \sigma^2_W/\sigma^{2,n}} \tag{3.24}$$

$$= (\beta^n)^{-1} - (\beta^n + \beta^W)^{-1}. \tag{3.25}$$

Equations (3.24) and (3.25) come directly from (3.23) and (3.20), using either variances or precisions.

Just as we let $\mathit{Var}^n[\cdot]$ be the variance given what we know after n measurements, let $\mathbb{E}^n$ be the expectation given what we know after n measurements. That is, if $W^1, \ldots, W^n$ are the first n measurements, we can write

$$\mathbb{E}^n \bar{\mu}^{n+1} \equiv \mathbb{E}(\bar{\mu}^{n+1} | W^1, \ldots, W^n) = \bar{\mu}^n.$$

We note in passing that $\mathbb{E}\bar{\mu}^{n+1}$ refers to the expectation before we have made any measurements, which means $W^1, \ldots, W^n$ are all random, as is W^{n+1}. By contrast, when we compute $\mathbb{E}^n \bar{\mu}^{n+1}$, $W^1, \ldots, W^n$ are assumed fixed, and only W^{n+1} is random. By the same token, $\mathbb{E}^{n+1}\bar{\mu}^{n+1} = \bar{\mu}^{n+1}$, where $\bar{\mu}^{n+1}$ is some number which is fixed because we assume that we already know $W^1, \ldots, W^{n+1}$. It is important to realize that when we write an expectation, we have to be explicit about what we are conditioning on. In practice, conditioning on history occurs implicitly when we collect a sequence of observations. In particular, if we have made n observations and have not yet made the $(n+1)$st observation, then $W^1, \ldots, W^n$ is known and W^{n+1} is random.

Using this property, we can write $\bar{\mu}^{n+1}$ in a different way that brings out the role of $\tilde{\sigma}^n$. Assume that we have made n observations and let

$$Z = \frac{\bar{\mu}^{n+1} - \bar{\mu}^n}{\tilde{\sigma}^n}.$$

We note that Z is a random variable only because we have not yet observed W^{n+1}. Normally we would index $Z = Z^{n+1}$ since it is a random variable that depends on W^{n+1}, but we are going to leave this indexing implicit. It is easy to see that $\mathbb{E}^n Z = 0$ and $Var^n Z = 1$. Also, since W^{n+1} is normally distributed, then Z is normally distributed, which means $Z \sim N(0,1)$. This means that we can write

$$\bar{\mu}^{n+1} = \bar{\mu}^n + \tilde{\sigma}^n Z. \tag{3.26}$$

Equation (3.26) makes it clear how $\bar{\mu}^n$ evolves over the observations. It also reinforces the idea that $\tilde{\sigma}^n$ is the change in the variance due to a single observation.

Equations (3.19) and (3.20) are the Bayesian counterparts of (3.17) and (3.18), although we have simplified the problem a bit by assuming that the variance of W is known. The belief state in the Bayesian view (with normally distributed beliefs) is given by the belief state

$$B^n = (\bar{\mu}^n, \beta^n).$$

If our prior belief about μ is normally distributed with mean $\bar{\mu}^n$ and precision β^n, and if W is normally distributed, then our posterior belief after $n+1$ observations is also normally distributed with mean $\bar{\mu}^{n+1}$ and precision β^{n+1}. We often use the term *Gaussian prior*, when we want to say that our prior is normally distributed. We also allow ourselves a slight abuse of notation: we use $\mathcal{N}\left(\mu, \sigma^2\right)$ to mean a normal distribution with mean μ and variance σ^2, but we also use the notation $\mathcal{N}\left(\mu, \beta\right)$ when we want to emphasize the precision instead of the variance.

Needless to say, it is especially convenient if the prior distribution and the posterior distribution are of the same basic type. When this is the case, we say that the prior and posterior are *conjugate*. This happens in a few special cases when the prior distribution and the distribution of W are chosen in a specific way. When this is the case, we say that the prior distribution is a *conjugate family*. The property that the posterior distribution is in the same family as the prior distribution is called *conjugacy*. The normal distribution is unusual in that the conjugate family is the same as the sampling family (the distribution of the measurement W is also normal). For this reason, this class of models is sometimes referred to as the "normal-normal" model (this phraseology becomes clearer below when we discuss other combinations).

In some cases, we may impose conjugacy as an approximation. For example, it might be the case that the prior distribution on μ is normal, but the distribution of the observation W is not normal (for example, it might be nonnegative). In this case, the posterior may not even have a convenient analytical form. But we might feel comfortable approximating the posterior as a normal distribution, in which case we would simply use (3.19)-(3.20) to update the mean and variance and then assume that the posterior distribution is normal.

3.4.2 Updating for correlated beliefs

We are now going to make the transition that instead of one number μ, we now have a vector $\mu_{x_1}, \mu_{x_2}, \ldots, \mu_{x_M}$. We can think of an element of μ as μ_x, which might be our estimate of a function $\mathbb{E} f(x, W)$ at x. Often, μ_x and $\mu_{x'}$ are correlated, as might happen when x is continuous, and x and x' are close to each other. But there are a number of examples that exhibit what we call *correlated beliefs*:

■ **EXAMPLE 3.1**

We are interested in finding the price of a product that maximizes total revenue. We believe that the function $R(p)$ that relates revenue to price is continuous. Assume that we set a price p^n and observe revenue R^{n+1} that is higher than we had expected. If we raise our estimate of the function $R(p)$ at the price p^n, our beliefs about the revenue at nearby prices should be higher.

■ **EXAMPLE 3.2**

We choose five people for the starting lineup of our basketball team and observe total scoring for one period. We are trying to decide if this group of five people is better than another lineup that includes three from the same group with two different people. If the scoring of these five people is higher than we had expected, we would probably raise our belief about the other group, since there are three people in common.

■ **EXAMPLE 3.3**

A physician is trying to treat diabetes using a treatment of three drugs, where she observes the drop in blood sugar from a course of a particular treatment. If one treatment produces a better-than-expected response, this would also increase our belief of the response from other treatments that have one or two drugs in common.

■ **EXAMPLE 3.4**

We are trying to find the highest concentration of a virus in the population. If the concentration of one group of people is higher than expected, our belief about other groups that are close (either geographically, or due to other relationships) would also be higher.

Correlated beliefs are a particularly powerful device in learning functions, allowing us to generalize the results of a single observation to other alternatives that we have not directly measured.

Let $\bar{\mu}_x^n$ be our belief about alternative x after n measurements. Now let

$$Cov^n(\mu_x, \mu_y) \quad = \quad \text{the covariance in our belief about } \mu_x \text{ and } \mu_y.$$

We let Σ^n be the covariance matrix, with element $\Sigma^n_{xy} = Cov^n(\mu_x, \mu_y)$. Just as we defined the precision β_x^n to be the reciprocal of the variance, we are going to define the precision matrix B^n to be

$$B^n = (\Sigma^n)^{-1}.$$

Let e_x be a column vector of zeroes with a 1 for element x, and as before we let W^{n+1} be the (scalar) observation when we decide to measure alternative x. We could label W^{n+1} as W_x^{n+1} to make the dependence on the alternative more explicit. For this discussion, we are going to use the notation that we choose to measure x^n and the resulting observation is W^{n+1}. If we choose to measure x^n, we can also interpret the observation as a column vector given by $W^{n+1}e_{x^n}$. Keeping in mind that $\bar{\mu}^n$ is a column vector of our beliefs about the expectation of μ, the Bayesian equation for updating this vector in the presence of correlated beliefs is given by

$$\bar{\mu}^{n+1} \quad = \quad (B^{n+1})^{-1}\left(B^n\bar{\mu}^n + \beta^W W^{n+1}e_{x^n}\right), \tag{3.27}$$

where B^{n+1} is given by

$$B^{n+1} \quad = \quad (B^n + \beta^W e_{x^n}(e_{x^n})^T). \tag{3.28}$$

Note that $e_x(e_x)^T$ is a matrix of zeroes with a one in row x, column x, whereas β^W is a scalar giving the precision of our measurement W.

It is possible to perform these updates without having to deal with the inverse of the covariance matrix. This is done using a result known as the Sherman-Morrison formula. If A is an invertible matrix (such as Σ^n) and u is a column vector (such as e_x), the Sherman-Morrison formula is

$$[A + uu^T]^{-1} = A^{-1} - \frac{A^{-1}uu^TA^{-1}}{1 + u^TA^{-1}u}. \tag{3.29}$$

Let $\lambda^W = \sigma_W^2 = 1/\beta^W$ be the variance of our measurement W^{n+1}. We are going to simplify our notation by assuming that our measurement variance is the same across all alternatives x, but if this is not the case, we can replace λ^W with λ_x^W throughout. Using the Sherman-Morrison formula, and letting $x = x^n$, we can rewrite the updating equations as

$$\bar{\mu}^{n+1}(x) \quad = \quad \bar{\mu}^n + \frac{W^{n+1} - \bar{\mu}_x^n}{\lambda^W + \Sigma_{xx}^n}\Sigma^n e_x, \tag{3.30}$$

$$\Sigma^{n+1}(x) \quad = \quad \Sigma^n - \frac{\Sigma^n e_x(e_x)^T\Sigma^n}{\lambda^W + \Sigma_{xx}^n}. \tag{3.31}$$

where we express the dependence of $\bar{\mu}^{n+1}(x)$ and $\Sigma^{n+1}(x)$ on the alternative x which we have chosen to measure.

To illustrate, assume that we have three alternatives with mean vector

$$\bar{\mu}^n = \begin{bmatrix} 20 \\ 16 \\ 22 \end{bmatrix}.$$

Assume that $\lambda^W = 9$ and that our covariance matrix Σ^n is given by

$$\Sigma^n = \begin{bmatrix} 12 & 6 & 3 \\ 6 & 7 & 4 \\ 3 & 4 & 15 \end{bmatrix}.$$

Assume that we choose to measure $x = 3$ and observe $W^{n+1} = W_3^{n+1} = 19$. Applying equation (3.30), we update the means of our beliefs using

$$
\begin{aligned}
\bar{\mu}^{n+1}(3) &= \begin{bmatrix} 20 \\ 16 \\ 22 \end{bmatrix} + \frac{19-22}{9+15} \begin{bmatrix} 12 & 6 & 3 \\ 6 & 7 & 4 \\ 3 & 4 & 15 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} 20 \\ 16 \\ 22 \end{bmatrix} + \frac{-3}{24} \begin{bmatrix} 3 \\ 4 \\ 15 \end{bmatrix} \\
&= \begin{bmatrix} 19.625 \\ 15.500 \\ 20.125 \end{bmatrix}.
\end{aligned}
$$

The update of the covariance matrix is computed using

$$
\begin{aligned}
\Sigma^{n+1}(3) &= \begin{bmatrix} 12 & 6 & 3 \\ 6 & 7 & 4 \\ 3 & 4 & 15 \end{bmatrix} - \frac{\begin{bmatrix} 12 & 6 & 3 \\ 6 & 7 & 4 \\ 3 & 4 & 15 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} [0 \ 0 \ 1] \begin{bmatrix} 12 & 6 & 3 \\ 6 & 7 & 4 \\ 3 & 4 & 15 \end{bmatrix}}{9+15} \\
&= \begin{bmatrix} 12 & 6 & 3 \\ 6 & 7 & 4 \\ 3 & 4 & 15 \end{bmatrix} - \frac{1}{24} \begin{bmatrix} 3 \\ 4 \\ 15 \end{bmatrix} [3 \ 4 \ 15] \\
&= \begin{bmatrix} 12 & 6 & 3 \\ 6 & 7 & 4 \\ 3 & 4 & 15 \end{bmatrix} - \frac{1}{24} \begin{bmatrix} 9 & 12 & 45 \\ 12 & 16 & 60 \\ 45 & 60 & 225 \end{bmatrix} \\
&= \begin{bmatrix} 12 & 6 & 3 \\ 6 & 7 & 4 \\ 3 & 4 & 15 \end{bmatrix} - \begin{bmatrix} 0.375 & 0.500 & 1.875 \\ 0.500 & 0.667 & 2.500 \\ 1.875 & 2.500 & 9.375 \end{bmatrix} \\
&= \begin{bmatrix} 11.625 & 5.500 & 1.125 \\ 5.500 & 6.333 & 1.500 \\ 1.125 & 1.500 & 5.625 \end{bmatrix}.
\end{aligned}
$$

These calculations are fairly easy, which means we can execute them even if we have thousands of alternatives. But we will run up against the limits of computer memory if the number of alternatives is in the 10^5 range or more, which arises when we consider problems where an alternative x is itself a multidimensional vector.

3.4.3 Gaussian process regression

A common strategy for approximating continuous functions is to discretize them, and then capture continuity by noting that the value of nearby points will be correlated, simply because of continuity. This is known as *Gaussian process regression*.

Assume that we have an unknown function $f(x)$ that is continuous in x which for the moment we will assume is a scalar that is discretized into the values $(x_1, x_2, \dots, x_M)$. Let $\bar{\mu}^n(x)$ be our estimate of $f(x)$ over our discrete set. Let $\mu(x)$ be the true value of $f(x)$ which, with our Bayesian hat on, we will interpret as a normally distributed random variable with mean $\bar{\mu}_x^0$ and variance $(\sigma_x^0)^2$ (this is our prior). We will further assume that

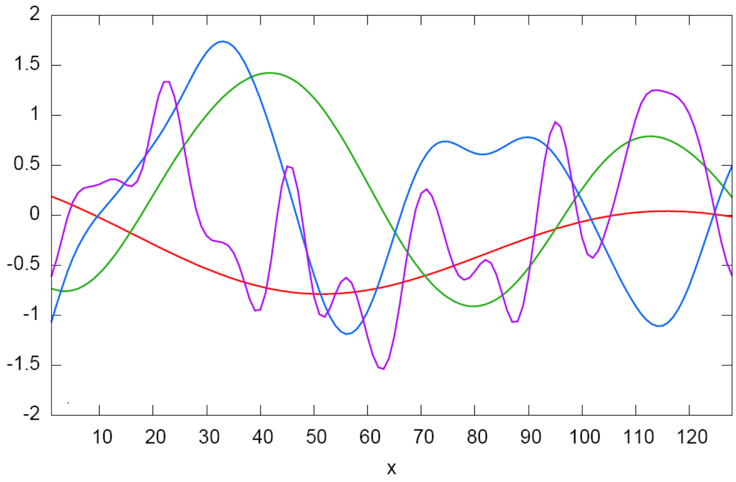


Figure 3.1 Illustration of a series of functions generated using Gaussian process regression (correlated beliefs) for different values of α.

μ_x and $\mu_{x'}$ are correlated with covariance

$$Cov(\mu_x, \mu_{x'}) = (\sigma^0)^2 e^{\alpha \|x-x'\|}, \tag{3.32}$$

where $\|x - x'\|$ is some distance metric such as $|x - x'|$ or $(x - x')^2$ (if x is a scalar) or $\sqrt{\sum_{i=1}^{I}(x_i - x_i')^2}$ if x is a vector. If $x = x'$ then we just pick up the variance in our belief about μ_x. The parameter α captures the degree to which x and x' are related as they get further apart.

Figure 3.1 illustrates a series of curves randomly generated from a belief model using the covariance function given in equation (3.32) for different values of α. Smaller values of α produce smoother curves with smaller undulations, because a smaller α translates to a higher covariance between more distant values of x and x'. As α increases, the covariance drops off and two different points on the curve become more independent.

Gaussian process regression (often shortened to just “GPR”) is a powerful approach for approximating low-dimensional functions that are continuous but otherwise have no specific structure. We present GPR here as a generalization of lookup table belief models, but it can also be characterized as a form of nonparametric statistics which we discuss below. In chapter 7 we will show how using GPR as a belief model can dramatically accelerate optimizing functions of continuous parameters such as drug dosages for medical applications, or the choice of temperature, pressure and concentration in a laboratory science application.

3.5 COMPUTING BIAS AND VARIANCE

Before we present our methods for hierarchical aggregation, we are going to need some basic results on bias and variance in statistical estimation. Assume we are trying to estimate a true but unknown parameter μ which we can observe, but we have to deal with both bias β and noise ε, which we write as

$$\hat{\mu}^n = \mu + \beta + \varepsilon^n. \tag{3.33}$$

Both μ and β are unknown, but we are going to assume that we have some way to make a noisy estimate of the bias that we are going to call $\hat{\beta}^n$. Later we are going to provide examples of how to get estimates of β.

Now let $\bar{\mu}^n$ be our estimate of μ after n observations. We will use the following recursive formula for $\bar{\mu}^n$

$$\bar{\mu}^n = (1-\alpha_{n-1})\bar{\mu}^{n-1} + \alpha_{n-1}\hat{\mu}^n.$$

We are interested in estimating the variance of $\bar{\mu}^n$ and its bias $\bar{\beta}^n$. We start by computing the variance of $\bar{\mu}^n$. We assume that our observations of μ can be represented using equation (3.33), where $\mathbb{E}\varepsilon^n = 0$ and $Var[\varepsilon^n] = \sigma^2$. With this model, we can compute the variance of $\bar{\mu}^n$ using

$$Var[\bar{\mu}^n] = \lambda^n \sigma^2, \tag{3.34}$$

where λ^n can be computed from the simple recursion

$$\lambda^n = \begin{cases} (\alpha_{n-1})^2, & n = 1, \\ (1-\alpha_{n-1})^2\lambda^{n-1} + (\alpha_{n-1})^2, & n > 1. \end{cases} \tag{3.35}$$

To see this, we start with $n = 1$. For a given (deterministic) initial estimate $\bar{\mu}^0$, we first observe that the variance of $\bar{\mu}^1$ is given by

$$\begin{aligned} Var[\bar{\mu}^1] &= Var[(1-\alpha_0)\bar{\mu}^0 + \alpha_0\hat{\mu}^1] \\ &= (\alpha_0)^2 Var[\hat{\mu}^1] \\ &= (\alpha_0)^2\sigma^2. \end{aligned}$$

For $\bar{\mu}^n$ for $n > 1$, we use a proof by induction. Assume that $Var[\bar{\mu}^{n-1}] = \lambda^{n-1}\sigma^2$. Then, since $\bar{\mu}^{n-1}$ and $\hat{\mu}^n$ are independent, we find

$$\begin{aligned} Var[\bar{\mu}^n] &= Var\left[(1-\alpha_{n-1})\bar{\mu}^{n-1} + \alpha_{n-1}\hat{\mu}^n\right] \\ &= (1-\alpha_{n-1})^2 Var\left[\bar{\mu}^{n-1}\right] + (\alpha_{n-1})^2 Var[\hat{\mu}^n] \\ &= (1-\alpha_{n-1})^2\lambda^{n-1}\sigma^2 + (\alpha_{n-1})^2\sigma^2 \qquad (3.36) \\ &= \lambda^n\sigma^2. \qquad (3.37) \end{aligned}$$

Equation (3.36) is true by assumption (in our induction proof), while equation (3.37) establishes the recursion in equation (3.35). This gives us the variance, assuming of course that σ^2 is known.

Using our assumption that we have access to a noisy estimate of the bias given by β^n, we can compute the mean-squared error using

$$\mathbb{E}\left[\left(\bar{\mu}^{n-1} - \bar{\mu}^n\right)^2\right] = \lambda^{n-1}\sigma^2 + (\beta^n)^2. \tag{3.38}$$

See exercise 5.5 to prove this. This formula gives the variance around the known mean, $\bar{\mu}^n$. For our purposes, it is also useful to have the variance around the observations $\hat{\mu}^n$. Let

$$\nu^n = \mathbb{E}\left[\left(\bar{\mu}^{n-1} - \hat{\mu}^n\right)^2\right]$$

be the mean squared error (including noise and bias) between the current estimate $\bar{\mu}^{n-1}$ and the observation $\hat{\mu}^n$. It is possible to show that (see exercise 5.6)

$$\nu^n = (1+\lambda^{n-1})\sigma^2 + (\beta^n)^2, \tag{3.39}$$

where λ^n is computed using (3.35).

In practice, we do not know σ^2, and we certainly do not know the bias β. As a result, we have to estimate both parameters from our data. We begin by providing an estimate of the bias using

$$\bar{\beta}^n = (1-\eta_{n-1})\bar{\beta}^{n-1} + \eta_{n-1}\beta^n,$$

where η_{n-1} is a (typically simple) stepsize rule used for estimating the bias and variance. As a general rule, we should pick a stepsize for η_{n-1} which produces larger stepsizes than α_{n-1} because we are more interested in tracking the true signal than producing an estimate with a low variance. We have found that a constant stepsize such as .10 works quite well on a wide range of problems, but if precise convergence is needed, it is necessary to use a rule where the stepsize goes to zero such as the harmonic stepsize rule (equation (6.12)).

To estimate the variance, we begin by finding an estimate of the total variation ν^n. Let $\bar{\nu}^n$ be the estimate of the total variance which we might compute using

$$\bar{\nu}^n = (1-\eta_{n-1})\bar{\nu}^{n-1} + \eta_{n-1}(\bar{\mu}^{n-1} - \hat{\mu}^n)^2.$$

Using $\bar{\nu}^n$ as our estimate of the total variance, we can compute an estimate of σ^2 using

$$(\bar{\sigma}^n)^2 = \frac{\bar{\nu}^n - (\bar{\beta}^n)^2}{1+\lambda^{n-1}}.$$

We can use $(\bar{\sigma}^n)^2$ in equation (3.34) to obtain an estimate of the variance of $\bar{\mu}^n$.

If we are doing true averaging (as would occur if we use a stepsize of $1/n$), we can get a more precise estimate of the variance for small samples by using the recursive form of the small sample formula for the variance

$$(\hat{\sigma}^2)^n = \frac{n-2}{n-1}(\hat{\sigma}^2)^{n-1} + \frac{1}{n}(\bar{\mu}^{n-1} - \hat{\mu}^n)^2. \tag{3.40}$$

The quantity $(\hat{\sigma}^2)^n$ is an estimate of the variance of $\hat{\mu}^n$. The variance of our estimate $\bar{\mu}^n$ is computed using

$$(\bar{\sigma}^2)^n = \frac{1}{n}(\hat{\sigma}^2)^n.$$

We are going to draw on these results in two settings, which are both distinguished by how estimates of the bias β^n are computed:

Hierarchical aggregation We are going to estimate a function at different levels of aggregation. We can assume that the estimate of the function at the most disaggregate level is noisy but unbiased, and then let the difference between the function at some level of aggregation and the function at the most disaggregate level as an estimate of the bias.

Transient functions Later, we are going to use these results to approximate value functions. It is the nature of algorithms for estimating value functions that the underlying process varies over time (see see this most clearly in chapter 14). In this setting, we are making observations from a truth that is changing over time, which introduces a bias.

3.6 LOOKUP TABLES AND AGGREGATION

Lookup table representations are the simplest and most general way to represent a function. If we are trying to model a function $f(x) = \mathbb{E}F(x, W)$, or perhaps a value function $V_t(S_t)$, assume that our function is defined over a discrete set of values $x_1, \ldots, x_M$ (or discrete states $\mathcal{S} = \{1, 2, \ldots, |\mathcal{S}|\}$). We wish to use observations of our function, whether they be $f^n = F(x^n, W^{n+1})$ (or $\hat{v}_t^n$, derived from simulations of the value of being in a state S_t), to create an estimate $\overline{F}^{n+1}$ (or $\overline{V}_t^{n+1}(S_t)$).

The problem with lookup table representations is that if our variable x (or state S) is a vector, then the number of possible values becomes exponentially larger with the number of dimensions. This is the classic curse of dimensionality. One strategy for overcoming the curse of dimensionality is to use aggregation, but picking a single level of aggregation is generally never satisfactory. In particular, we typically have to start with no data, and steadily build up an estimate of a function.

We can accomplish this transition from little to no data, to increasing numbers of observations, by using hierarchical aggregation. Instead of picking a single level of aggregation, we work with a family of aggregations which are hierarchically structured.

3.6.1 Hierarchical aggregation

Lookup table representations of functions often represent the first strategy we consider because it does not require that we assume any structural form. The problem is that it requires that we discretize the domain $\mathcal{X}$ of the function $f(x)$. If $x = x_1, \ldots, x_d$ is multidimensional, then the number of elements in $\mathcal{X}$ grows exponentially with the number of dimensions, often limiting the number of dimensions to around three. If the elements x_d are continuous, we can limit the growth by using a coarse discretization, but this introduces discretization errors (a topic that has received considerable attention in the literature on functional approximations). Even more important for adaptive learning algorithms is that this approach does not help with the transition from very few observations (in the early stages of an algorithm) to later stages, where we may actually have quite a few observations, at least in some regions of the function.

One powerful strategy that makes it possible to extend lookup tables is the use of hierarchical aggregation. Rather than simply aggregating a state space into a smaller space, we pose a family of aggregations, and then combine these based on the statistics of our estimates at each level of aggregation. This is not a panacea (nothing is), and should not be viewed as a method that "solves" the curse of dimensionality, but it does represent a powerful addition to our toolbox of approximation strategies.

We can illustrate hierarchical aggregation using our nomadic trucker example that we first introduced in section 2.3.3.3. In this setting, we are managing a truck driver who is picking up and dropping off loads (imagine taxicabs for freight), where the driver has to choose loads based on both how much money he will make moving the load, and the value of landing at the destination of the load. Complicating the problem is that the driver is described by a multidimensional attribute vector $r = (r_1, r_2, \ldots, r_d)$ which includes attributes such as the location of a truck (which means location in a region), his equipment type (the type of trailer he is hauling, such as a regular "van" for freight, a refrigerated van or a flatbed), and his home location (again, a region).

If our nomadic trucker is described by the state vector $S_t = r_t$ which we act on with an action a_t (moving one of the available loads), the transition function $S_{t+1} =$

$S^M(S_t, a_t, W_{t+1})$ may represent the state vector at a high level of detail (some values may be continuous). But the decision problem

$$\max_{a_t \in \mathcal{A}} \left(C(S_t, a_t) + \gamma \mathbb{E}\{\overline{V}_{t+1}(G(S_{t+1}))|S_t\} \right) \tag{3.41}$$

uses a value function $\overline{V}_{t+1}(G(S_{t+1}))$, where $G(\cdot)$ is an aggregation function that maps the original (and very detailed) state S into something much simpler. The aggregation function G may ignore a dimension, discretize it, or use any of a variety of ways to reduce the number of possible values of a state vector. This also reduces the number of parameters we have to estimate. In what follows, we drop the explicit reference of the aggregation function G and simply use $\overline{V}_{t+1}(S_{t+1})$. The aggregation is implicit in the value function approximation.

Some examples of aggregation include:

Spatial - A transportation company is interested in estimating the value of truck drivers at a particular location. Locations may be calculated at the level of a five-digit zip code (there are about 55,000 in the United States), three-digit zip code (about 1,000), or the state level (48 contiguous states).

Temporal - A bank may be interested in estimating the value of holding an asset at a point in time. Time may be measured by the day, week, month, or quarter.

Continuous parameters - The state of an aircraft may be its fuel level; the state of a traveling salesman may be how long he has been away from home; the state of a water reservoir may be the depth of the water; the state of the cash reserve of a mutual fund is the amount of cash on hand at the end of the day. These are examples of systems with at least one dimension of the state that is at least approximately continuous. The variables may all be discretized into intervals of varying lengths.

Hierarchical classification - A portfolio problem may need to estimate the value of investing money in the stock of a particular company. It may be useful to aggregate companies by industry segment (for example, a particular company might be in the chemical industry, and it might be further aggregated based on whether it is viewed as a domestic or multinational company). Similarly, problems of managing large inventories of parts (for cars, for example) may benefit by organizing parts into part families (transmission parts, engine parts, dashboard parts).

The examples below provide additional illustrations.

■ **EXAMPLE 3.1**

The state of a jet aircraft may be characterized by multiple attributes which include spatial and temporal dimensions (location and flying time since the last maintenance check), as well other attributes. A continuous parameter could be the fuel level, an attribute that lends itself to hierarchical aggregation might be the specific type of aircraft. We can reduce the number of states (attributes) of this resource by aggregating each dimension into a smaller number of potential outcomes.

Aggregation level	Location	Fleet type	Domicile	Size of state space
0	Sub-region	Fleet	Region	$400 \times 5 \times 100 = 200,000$
1	Region	Fleet	Region	$100 \times 5 \times 100 = 50,000$
2	Region	Fleet	Zone	$100 \times 5 \times 10 = 5,000$
3	Region	Fleet	-	$100 \times 5 \times 1 = 500$
4	Zone	-	-	$10 \times 1 \times 1 = 10$

Table 3.1 Examples of aggregations of the state space for the nomadic trucker problem. '-' indicates that the particular dimension is ignored.

■ **EXAMPLE 3.2**

The state of a portfolio might consist of the number of bonds which are characterized by the source of the bond (a company, a municipality or the federal government), the maturity (six months, 12 months, 24 months), when it was purchased, and its rating by bond agencies. Companies can be aggregated up by industry segment. Bonds can be further aggregated by their bond rating.

■ **EXAMPLE 3.3**

Blood stored in blood banks can be characterized by type, the source (which might indicate risks for diseases), age (it can be stored for up to 42 days), and the current location where it is being stored. A national blood management agency might want to aggregate the state space by ignoring the source (ignoring a dimension is a form of aggregation), discretizing the age from days into weeks, and aggregating locations into more aggregate regions.

■ **EXAMPLE 3.4**

The value of an asset is determined by its current price, which is continuous. We can estimate the asset using a price discretized to the nearest dollar.

There are many applications where aggregation is naturally hierarchical. For example, in our nomadic trucker problem we might want to estimate the value of a truck based on three attributes: location, home domicile, and fleet type. The first two represent geographical locations, which can be represented (for this example) at three levels of aggregation: 400 sub-regions, 100 regions, and 10 zones. Table 3.1 illustrates five levels of aggregation that might be used. In this example, each higher level can be represented as an aggregation of the previous level.

Aggregation is also useful for continuous variables. Assume that our state variable is the amount of cash we have on hand, a number that might be as large as \$10 million dollars. We might discretize our state space in units of \$1 million, \$100 thousand, \$10 thousand,

\$1,000, \$100, and \$10. This discretization produces a natural hierarchy since 10 segments at one level of aggregation naturally group into one segment at the next level of aggregation.

Hierarchical aggregation is often the simplest to work with, but in most cases there is no reason to assume that the structure is hierarchical. In fact, we may even use overlapping aggregations (sometimes known as "soft" aggregation), where the same state s aggregates into multiple elements in $\mathcal{S}^g$. For example, assume that s represents an (x, y) coordinate in a continuous space which has been discretized into the set of points $(x_i, y_i)_{i \in \mathcal{I}}$. Further assume that we have a distance metric $\rho((x, y), (x_i, y_i))$ that measures the distance from any point (x, y) to every aggregated point (x_i, y_i) , $i \in \mathcal{I}$. We might use an observation at the point (x, y) to update estimates at each (x_i, y_i) with a weight that declines with $\rho((x, y), (x_i, y_i))$.

3.6.2 Estimates of different levels of aggregation

Assume we are trying to approximate a function $f(x)$, $x \in \mathcal{X}$. We begin by defining a family of aggregation functions

$$G^g : \mathcal{X} \rightarrow \mathcal{X}^{(g)}.$$

$\mathcal{X}^{(g)}$ represents the g^{th} level of aggregation of the domain $\mathcal{X}$. Let

$\mathcal{G}$ = The set of indices corresponding to the levels of aggregation.

In this section, we assume we have a single aggregation function G that maps the disaggregate state $x \in \mathcal{X} = \mathcal{X}^{(0)}$ into an aggregated space $\mathcal{X}^{(g)}$. In section 3.6.3, we let $g \in \mathcal{G} = \{0, 1, 2, \ldots\}$ and we work with all levels of aggregation at the same time.

To begin our study of aggregation, we first need to characterize how we sample values x at the disaggregate level. For this discussion, we assume we have two exogenous processes: At iteration n, the first process chooses a value to sample (which we denote by x^n), and the second produces an observation of the value of being in state

$$\hat{f}^n(x^n) = f(x^n) + \varepsilon^n.$$

Later, we are going to assume that x^n is determined by some policy, but for now, we can treat this as purely exogenous.

We need to characterize the errors that arise in our estimate of the function. Let

$f_x^{(g)}$ = The true estimate of the gth aggregation of the original function $f(x)$.

We assume that $f^{(0)}(x) = f(x)$, which means that the zeroth level of aggregation is the true function.

Let

$\bar{f}_x^{(g,n)}$ = The estimate of the value of $f(x)$ at the g^{th} level of aggregation after n observations.

Throughout our discussion, a bar over a variable means it was computed from sample observations. A hat means the variable was an exogenous observation.

When we are working at the most disaggregate level ($g = 0$), the state s that we measure is the observed state $s = \hat{s}^n$. For $g > 0$, the subscript x in $\bar{f}_x^{(g,n)}$ refers to $G^g(x^n)$, or the

g^{th} level of aggregation of $f(x)$ at $x = x^n$. Given an observation $(x^n, \hat{f}^n(x^n))$, we would update our estimate of the $f^{(g)}(x)$ using

$$\bar{f}_x^{(g,n)} = (1 - \alpha_{x,n-1}^{(g)})\bar{f}_x^{(g,n-1)} + \alpha_{x,n-1}^{(g)}\hat{f}^n(x).$$

Here, we have written the stepsize $\alpha_{x,n-1}^{(g)}$ to explicitly represent the dependence on the decision x and level of aggregation. Implicit is that this is also a function of the number of times that we have updated $\bar{f}_x^{(g,n)}$ by iteration n, rather than a function of n itself.

To illustrate, imagine that our nomadic trucker is described by the vector $x = (\text{Loc}, \text{Equip}, \text{Home}, \text{DOThrs}, \text{Days})$, where "Loc" is location, "Equip" denotes the type of trailer (long, short, refrigerated), "Home" is the location of where he lives, "DOThrs" is a vector giving the number of hours the driver has worked on each of the last eight days, and "Days" is the number of days the driver has been away from home. We are going to estimate the value $f(x)$ for different levels of aggregation of x, where we aggregate purely by ignoring certain dimensions of s. We start with our original disaggregate observation $\hat{f}(x)$, which we are going to write as

$$\hat{f}\begin{pmatrix}\text{Loc}\\ \text{Equip}\\ \text{Home}\\ \text{DOThrs}\\ \text{Days}\end{pmatrix} = f(x) + \varepsilon.$$

We now wish to use this estimate of the value of a driver with attribute x to produce value functions at different levels of aggregation. We can do this by simply smoothing this disaggregate estimate in with estimates at different levels of aggregation, as in

$$\begin{aligned}
\bar{v}^{(1,n)}\begin{pmatrix}\text{Loc}\\ \text{Equip}\\ \text{Home}\end{pmatrix} &= (1 - \alpha_{x,n-1}^{(1)})\bar{f}^{(1,n-1)}\begin{pmatrix}\text{Loc}\\ \text{Equip}\\ \text{Home}\end{pmatrix} + \alpha_{x,n-1}^{(1)}\hat{f}\begin{pmatrix}\text{Loc}\\ \text{Equip}\\ \text{Home}\\ \text{DOThrs}\\ \text{Days}\end{pmatrix},\\
\bar{v}^{(2,n)}\begin{pmatrix}\text{Loc}\\ \text{Equip}\end{pmatrix} &= (1 - \alpha_{x,n-1}^{(2)})\bar{f}^{(2,n-1)}\begin{pmatrix}\text{Loc}\\ \text{Equip}\end{pmatrix} + \alpha_{x,n-1}^{(2)}\hat{f}\begin{pmatrix}\text{Loc}\\ \text{Equip}\\ \text{Home}\\ \text{DOThrs}\\ \text{Days}\end{pmatrix},\\
\bar{v}^{(3,n)}\begin{pmatrix}\text{Loc}\end{pmatrix} &= (1 - \alpha_{x,n-1}^{(3)})\bar{f}^{(3,n-1)}\begin{pmatrix}\text{Loc}\end{pmatrix} + \alpha_{x,n-1}^{(3)}\hat{v}\begin{pmatrix}\text{Loc}\\ \text{Equip}\\ \text{Home}\\ \text{DOThrs}\\ \text{Days}\end{pmatrix}.
\end{aligned}$$

In the first equation, we are smoothing the value of a driver based on a five-dimensional state vector, given by x, in with an approximation indexed by a three-dimensional state vector. The second equation does the same using value function approximation indexed by a two-dimensional state vector, while the third equation does the same with a one-dimensional state vector. It is very important to keep in mind that the stepsize must reflect the number of times a state has been updated.

We need to estimate the variance of $\bar{f}_x^{(g,n)}$. Let

$(s_x^2)^{(g,n)}$ = The estimate of the variance of observations made of the function at x, using data from aggregation level g, after n observations.

$(s_x^2)^{(g,n)}$ is the estimate of the variance of the observations $\hat{f}$ when we observe the function at $x = x^n$ which aggregates to x (that is, $G^g(x^n) = x$). We are really interested in the variance of our estimate of the mean, $\bar{f}_x^{(g,n)}$. In section 3.5, we showed that

$$\begin{aligned}(\bar{\sigma}_x^2)^{(g,n)} &= Var[\bar{f}_x^{(g,n)}] \\ &= \lambda_x^{(g,n)}(s_x^2)^{(g,n)}, \end{aligned} \tag{3.42}$$

where $(s_x^2)^{(g,n)}$ is an estimate of the variance of the observations $\hat{f}^n$ at the g^{th} level of aggregation (computed below), and $\lambda_s^{(g,n)}$ can be computed from the recursion

$$\lambda_x^{(g,n)} = \begin{cases}(\alpha_{x,n-1}^{(g)})^2, & n = 1, \\ (1 - \alpha_{x,n-1}^{(g)})^2 \lambda_x^{(g,n-1)} + (\alpha_{x,n-1}^{(g)})^2, & n > 1.\end{cases}$$

Note that if the stepsize $\alpha_{x,n-1}^{(g)}$ goes to zero, then $\lambda_x^{(g,n)}$ will also go to zero, as will $(\bar{\sigma}_x^2)^{(g,n)}$. We now need to compute $(s_x^2)^{(g,n)}$ which is the estimate of the variance of observations $\hat{f}^n$ at points x^n for which $G^g(x^n) = x$ (the observations of states that aggregate up to x). Let $\bar{\nu}_x^{(g,n)}$ be the total variation, given by

$$\bar{\nu}_x^{(g,n)} = (1 - \eta_{n-1})\bar{\nu}_x^{(g,n-1)} + \eta_{n-1}(\bar{f}_x^{(g,n-1)} - \hat{f}_x^n)^2,$$

where η_{n-1} follows some stepsize rule (which may be just a constant). We refer to $\bar{\nu}_x^{(g,n)}$ as the total variation because it captures deviations that arise both due to measurement noise (the randomness when we compute $\hat{f}^n(x)$) and bias (since $\bar{f}_x^{(g,n-1)}$ is a biased estimate of the mean of $\hat{f}^n(x)$).

We finally need an estimate of the bias from aggregation which we find by computing

$$\bar{\beta}_x^{(g,n)} = \bar{f}_x^{(g,n)} - \bar{f}_x^{(0,n)}. \tag{3.43}$$

We can separate out the effect of bias to obtain an estimate of the variance of the error using

$$(s_x^2)^{(g,n)} = \frac{\bar{\nu}_x^{(g,n)} - (\bar{\beta}_x^{(g,n)})^2}{1 + \lambda^{n-1}}. \tag{3.44}$$

In the next section, we put the estimate of aggregation bias, $\bar{\beta}_x^{(g,n)}$, to work.

The relationships are illustrated in figure 3.2, which shows a simple function defined over a single, continuous state (for example, the price of an asset). If we select a particular state s, we find we have only two observations for that state, versus seven for that section of the function. If we use an aggregate approximation, we would produce a single number over that range of the function, creating a bias between the true function and the aggregated estimate. As the illustration shows, the size of the bias depends on the shape of the function in that region.

One method for choosing the best level of aggregation is to choose the level that minimizes $(\bar{\sigma}_s^2)^{(g,n)} + (\bar{\beta}_s^{(g,n)})^2$, which captures both bias and variance. In the next section, we use the bias and variance to develop a method that uses estimates at all levels of aggregation at the same time.

3.6.3 Combining multiple levels of aggregation

Rather than try to pick the best level of aggregation, it is intuitively appealing to use a weighted sum of estimates at different levels of aggregation. The simplest strategy is to use

$$\bar{f}_x^n = \sum_{g \in \mathcal{G}} w^{(g)} \bar{f}_x^{(g)}, \tag{3.45}$$

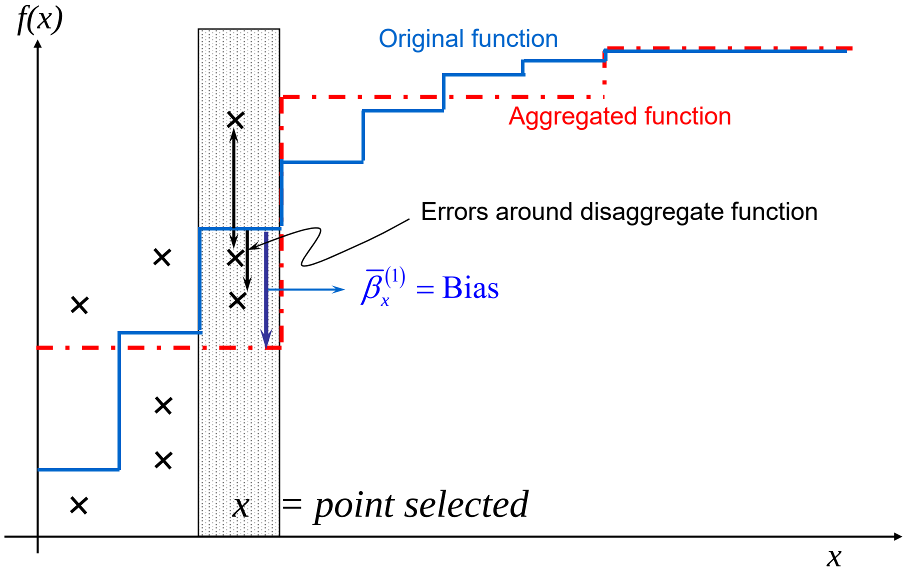

f(x)
Original function
Aggregated function
Errors around disaggregate function
$\bar{\beta}_x^{(1)}$ = Bias
x = point selected
x

Figure 3.2 Illustration of a disaggregate function, an aggregated approximation and a set of samples. For a particular state s, we show the estimate and the bias.

where $w^{(g)}$ is the weight applied to the g^{th} level of aggregation. We would expect the weights to be positive and sum to one, but we can also view these simply as coefficients in a regression function. In such a setting, we would normally write the regression as

$$\overline{F}(x|\theta) = \theta_0 + \sum_{g \in \mathcal{G}} \theta_g \bar{f}_x^{(g)},$$

(see section 18.4.1 for a discussion of general regression methods). The problem with this strategy is that the weight does not depend on the value of x. Intuitively, it makes sense to put a higher weight on points x which have more observations, or where the estimated variance is lower. This behavior is lost if the weight does not depend on x.

In practice, we will generally observe some states much more frequently than others, suggesting that the weights should depend on x. To accomplish this, we need to use

$$\bar{f}_x^n = \sum_{g \in \mathcal{G}} w_x^{(g)} \bar{f}_x^{(g,n)}.$$

Now the weight depends on the point being estimated, allowing us to put a higher weight on the disaggregate estimates when we have a lot of observations. This is clearly the most natural, but when the domain $\mathcal{X}$ is large, we face the challenge of computing thousands (perhaps hundreds of thousands) of weights. If we are going to go this route, we need a fairly simple method to compute the weights.

We can view the estimates $(\bar{f}^{(g,n)})_{g \in \mathcal{G}}$ as different ways of estimating the same quantity. There is an extensive statistics literature on this problem. For example, it is well known

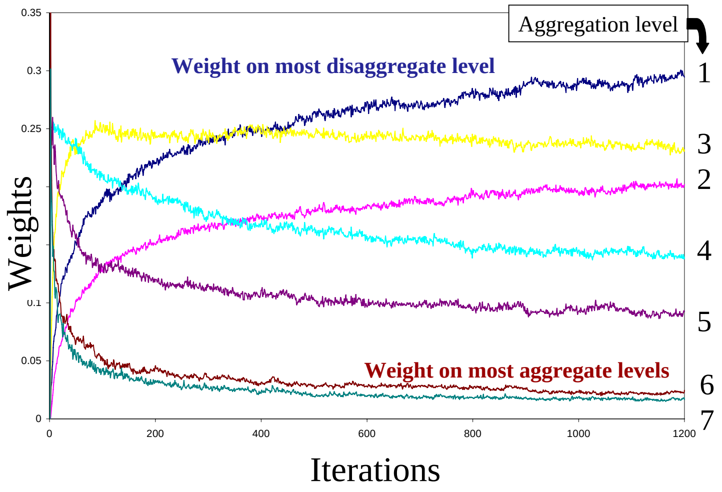


Figure 3.3 Average weight (across all states) for each level of aggregation using equation (3.47).

that the weights that minimize the variance of $\bar{f}^n_x$ in equation (3.45) are given by

$$w^{(g)}_x \;\propto\; \left((\bar{\sigma}^2_x)^{(g,n)}\right)^{-1}.$$

Since the weights should sum to one, we obtain

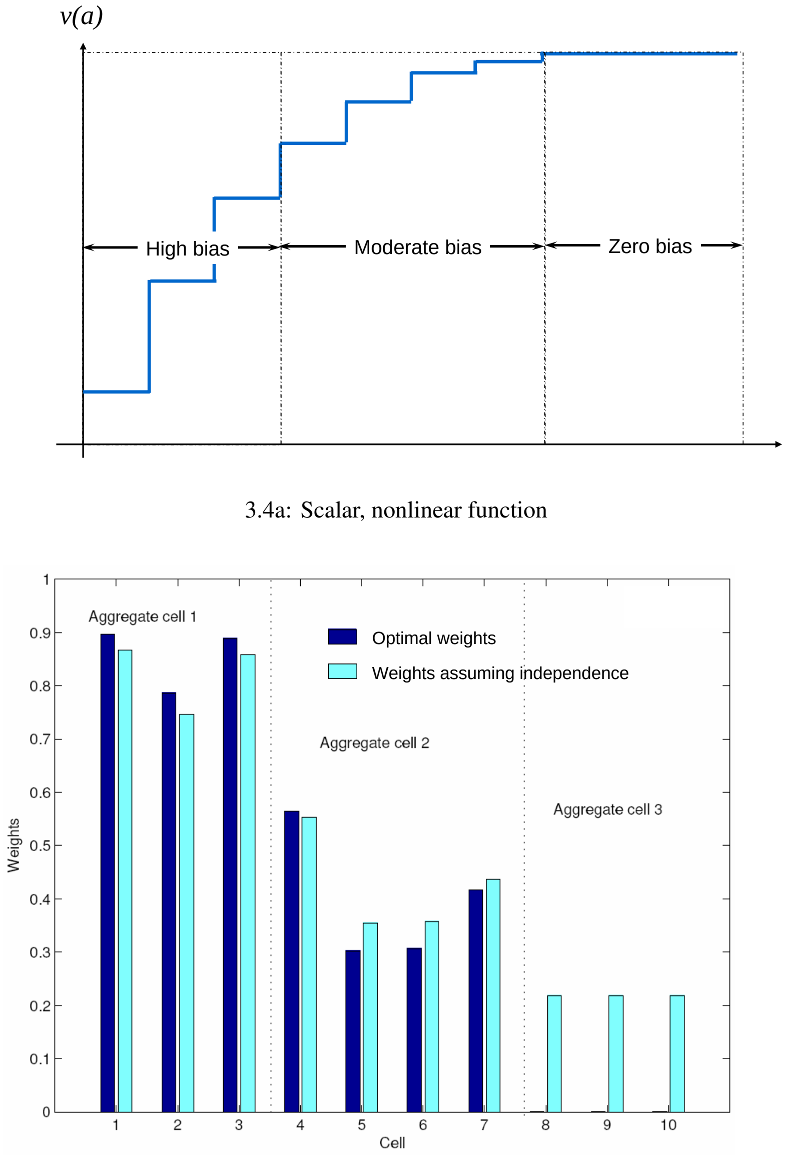
$$w^{(g)}_x = \left(\frac{1}{(\bar{\sigma}^2_x)^{(g,n)}}\right)\left(\sum_{g\in\mathcal{G}}\frac{1}{(\bar{\sigma}^2_x)^{(g,n)}}\right)^{-1}. \tag{3.46}$$

These weights work if the estimates are unbiased, which is clearly not the case. This is easily fixed by using the total variation (variance plus the square of the bias), producing the weights

$$w^{(g,n)}_x = \frac{1}{\left((\bar{\sigma}^2_x)^{(g,n)} + \left(\bar{\beta}^{(g,n)}_x\right)^2\right)}\left(\sum_{g'\in\mathcal{G}}\frac{1}{\left((\bar{\sigma}^2_x)^{(g',n)} + \left(\bar{\beta}^{(g',n)}_x\right)^2\right)}\right)^{-1}. \tag{3.47}$$

These are computed for each level of aggregation $g \in \mathcal{G}$. Furthermore, we compute a different set of weights for each point x. $(\bar{\sigma}^2_x)^{(g,n)}$ and $\bar{\beta}^{(g,n)}_x$ are easily computed recursively using equations (3.42) and (3.43), which makes the approach well suited to large scale applications. Note that if the stepsize used to smooth $\hat{f}^n$ goes to zero, then the variance $(\bar{\sigma}^2_x)^{(g,n)}$ will also go to zero as $n \rightarrow \infty$. However, the bias $\bar{\beta}^{(g,n)}_x$ will in general not go to zero.

Figure 3.3 shows the average weight put on each level of aggregation (when averaged over all the states s) for a particular application. The behavior illustrates the intuitive property that the weights on the aggregate level are highest when there are only a few observations, with a shift to the more disaggregate level as the algorithm progresses. This is a very important behavior when approximating functions recursively. It is simply not



3.4a: Scalar, nonlinear function

3.4b: Weight given to disaggregate level

Figure 3.4 The weight given to the disaggregate level for a two-level problem at each of 10 points, with and without the independence assumption (from George et al. (2008)).

possible to produce good function approximations with only a few data points, so it is important to use simple functions (with only a few parameters).

The weights computed using (3.3) minimize the variance in the estimate $\bar{f}_x^{(g)}$ if the estimates at different levels of aggregation are independent, but this is simply not going to be the case. $\bar{f}_x^{(0)}$ (an estimate of s at the most disaggregate level) and $\bar{f}_x^{(1)}$ will be correlated

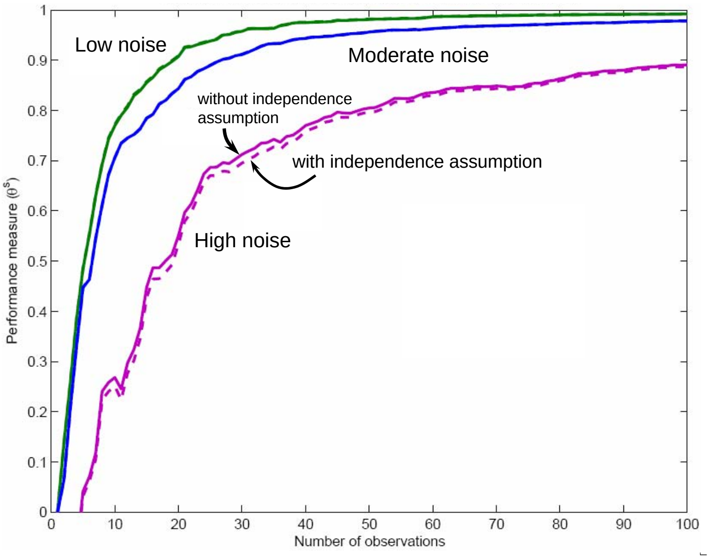


Figure 3.5 The effect of ignoring the correlation between estimates at different levels of aggregation

since $\bar{f}_x^{(1)}$ is estimated using some of the same observations used to estimate $\bar{f}_x^{(0)}$. So it is fair to ask if the weights produce accurate estimates.

To get a handle on this question, consider the scalar function in figure 3.4a. At the disaggregate level, the function is defined for 10 discrete values. This range is then divided into three larger intervals, and an aggregated function is created by estimating the function over each of these three larger ranges. Instead of using the weights computed using (3.47), we can fit a regression of the form

$$\hat{f}^n(x) = \theta_0 \bar{f}^{(0,n)}(x) + \theta_1 \bar{f}^{(1,n)}(x). \tag{3.48}$$

The parameters θ_0 and θ_1 can be fit using regression techniques. Note that while we would expect $\theta_0 + \theta_1$ to be approximately 1, there is no formal guarantee of this. If we use only two levels of aggregation, we can find (θ_0, θ_1) using linear regression and can compare these weights to those computed using equation (3.47) where we assume independence.

For this example, the weights are shown in figure 3.4b. The figure illustrates that the weight on the disaggregate level is highest when the function has the greatest slope, which produces the highest biases. When we compute the optimal weights (which captures the correlation), the weight on the disaggregate level for the portion of the curve that is flat is zero, as we would expect. Note that when we assume independence, the weight on the disaggregate level (when the slope is zero) is no longer zero. Clearly a weight of zero is best because it means that we are aggregating all the points over the interval into a single estimate, which is going to be better than trying to produce three individual estimates.

One would expect that using the optimal weights, which captures the correlations between estimates at different levels of aggregation, would also produce better estimates of the function itself. This does not appear to be the case. We compared the errors between the estimated function and the actual function using both methods for computing weights, using three levels of noise around the function. The results are shown in figure 3.5, which indicates that there is virtually no difference in the accuracy of the estimates produced by the two methods. This observation has held up under a number of experiments.

3.7 LINEAR PARAMETRIC MODELS

Up to now, we have focused on lookup-table representations of functions, where if we are at a point x (or state s), we compute an approximation $\overline{F}(x)$ (or $\overline{V}(s)$) that is an estimate of the function at x (or state s). Using aggregation (even mixtures of estimates at different levels of aggregation) is still a form of look-up table (we are just using a simpler lookup-table). Lookup tables offer tremendous flexibility, but generally do not scale to higher dimensional variables (x or s), and do not allow you to take advantage of structural relationships.

There has been considerable interest in estimating functions using regression methods. A classical presentation of linear regression poses the problem of estimating a parameter vector θ to fit a model that predicts a variable y using a set of observations (known as covariates in the machine learning community) $(x_i)_{i\in\mathcal{I}}$, where we assume a model of the form

$$y = \theta_0 + \sum_{i=1}^{I} \theta_i x_i + \varepsilon. \tag{3.49}$$

The variables x_i might be called independent variables, explanatory variables, or covariates, depending on the community. In dynamic programming where we want to estimate a value function $V^\pi(S_t)$, we might write

$$\overline{V}(S|\theta) = \sum_{f\in\mathcal{F}} \theta_f \phi_f(S),$$

where $(\phi_f(S))_{f\in\mathcal{F}}$ are known variously as *basis functions* or *features*, but are also referred to by names such as *covariates* or simply "independent variables." We might use this vocabulary regardless of whether we are approximating a value function or the policy itself. In fact, if we write our policy of the form

$$X^\pi(S_t|\theta) = \sum_{f\in\mathcal{F}} \theta_f \phi_f(S_t),$$

we would refer to $X^\pi(S_t|\theta)$ as an *affine policy* ("affine" is just a fancy name for linear, by which we mean linear in θ).

Linear models are arguably the most popular approximation strategy for complex problems because they handle high-dimensionality by imposing a linear structure (which also means separable and additive). Using this language, instead of an independent variable x_i, we would have a basis function $\phi_f(S)$, where $f \in \mathcal{F}$ is a *feature*. $\phi_f(S)$ might be an indicator variable (e.g., 1 if we have an 'X' in the center square of our tic-tac-toe board), a discrete number (the number of X's in the corners of our tic-tac-toe board), or a continuous quantity (the price of an asset, the amount of oil in our inventories, the amount of $AB-$ blood on hand at the hospital). Some problems might have fewer than 10 features; others may have dozens; and some may have hundreds of thousands. In general, however, we would write our value function in the form

$$\overline{V}(S|\theta) = \sum_{f\in\mathcal{F}} \theta_f \phi_f(S).$$

In a time dependent model, the parameter vector θ and basis functions ϕ may also be indexed by time (but not necessarily).

In the remainder of this section, we provide a brief review of linear regression, followed by some examples of regression models. We close with a more advanced presentation that provides insights into the geometry of basis functions (including a better understanding of why they are called "basis functions"). Given the tremendous amount of attention this class of approximations has received in the literature, we defer to chapter 17 a full description of how to fit linear regression models recursively for an ADP setting.

3.7.1 Linear regression review

Let y^n be the n^{th} observation of our dependent variable (what we are trying to predict) based on the observation $(x_1^n, x_2^n, \ldots, x_I^n)$ of our independent (or explanatory) variables (the x_i are equivalent to the basis functions we used earlier). Our goal is to estimate a parameter vector θ that solves

$$\min_\theta \sum_{m=1}^{n} \left(y^m - (\theta_0 + \sum_{i=1}^{I} \theta_i x_i^m) \right)^2 . \tag{3.50}$$

This is the standard linear regression problem.

Throughout this section, we assume that the underlying process from which the observations y^n are drawn is stationary (an assumption that is almost never satisfied in approximate dynamic programming).

If we define $x_0 = 1$, we let

$$x^n = \begin{pmatrix} x_0^n \\ x_1^n \\ \vdots \\ x_I^n \end{pmatrix}$$

be an $I+1$-dimensional column vector of observations. Throughout this section, and unlike the rest of the book, we use traditional vector operations, where $x^T x$ is an inner product (producing a scalar) while xx^T is an outer product, producing a matrix of cross terms.

Letting θ be the column vector of parameters, we can write our model as

$$y = \theta^T x + \varepsilon.$$

We assume that the errors $(\varepsilon^1, \ldots, \varepsilon^n)$ are independent and identically distributed. We do not know the parameter vector θ, so we replace it with an estimate $\bar{\theta}$ which gives us the predictive formula

$$\bar{y}^n = (\bar{\theta})^T x^n,$$

where $\bar{y}^n$ is our predictor of y^{n+1}. Our prediction error is

$$\hat{\varepsilon}^n = y^n - (\bar{\theta})^T x^n.$$

Our goal is to choose θ to minimize the mean squared error

$$\min_\theta \sum_{m=1}^{n} (y^m - \theta^T x^m)^2. \tag{3.51}$$

It is well known that this can be solved very simply. Let X^n be the n by $I+1$ matrix

$$X^n = \begin{pmatrix} x_0^1 & x_1^1 & & x_I^1 \\ x_0^2 & x_1^2 & & x_I^2 \\ \vdots & \vdots & \cdots & \vdots \\ x_0^n & x_1^n & & x_I^n \end{pmatrix}.$$

Next, denote the vector of observations of the dependent variable as

$$Y^n = \begin{pmatrix} y^1 \\ y^2 \\ \vdots \\ y^n \end{pmatrix}.$$

The optimal parameter vector $\bar{\theta}$ (after n observations) is given by

$$\bar{\theta} \quad = \quad [(X^n)^T X^n]^{-1}(X^n)^T Y^n. \tag{3.52}$$

Solving a static optimization problem such as (3.51), which produces the elegant equations for the optimal parameter vector in (3.52), is the most common approach taken by the statistics community. It has little direct application in approximate dynamic programming since our problems tend to be recursive in nature, reflecting the fact that at each iteration we obtain new observations, which require updates to the parameter vector. In addition, our observations tend to be notoriously nonstationary. Later, we show how to overcome this problem using the methods of recursive statistics.

3.7.2 Sparse additive models and Lasso

It is not hard to create models where there are a large number of explanatory variables. Some examples include:

■ **EXAMPLE 3.1**

A physician is trying to choose the best medical treatment for a patient, which may be described by thousands of different characteristics. It is unlikely that all of these characteristics have strong explanatory power.

■ **EXAMPLE 3.2**

A scientist is trying to design probes to identify the structure of RNA molecules. There are hundreds of locations where a probe can be attached. The challenge is to design probes to learn a statistical model that has hundreds of parameters (corresponding to each location).

■ **EXAMPLE 3.3**

An internet provider is trying to maximize ad-clicks, where each ad is characterized by an entire dataset consisting of all the text and graphics. A model can be created by generating hundreds of features based on word patterns within the ad. The problem is to learn which features are most important by carefully selecting ads.

In these settings, we are trying to approximate a function $f(S)$ where S is our "state variable" consisting of all the data (describing patients, the RNA molecule, or the features within an ad). $f(S)$ might be the response (medical successes or costs, or clicks on ads), which we approximate using

$$\overline{F}(S|\theta) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(S). \tag{3.53}$$

Now imagine that there are hundreds of features in the set $\mathcal{F}$, but we anticipate that $\theta_f = 0$ for many of these. In this case, we would view equation (3.53) as a *sparse additive* model, where the challenge is to identify a model with the highest explanatory power which means excluding the parameters which do not contribute very much.

Imagine we have a dataset consisting of $(f^n, S^n)_{n=1}^N$ where f^n is the observed response corresponding to the information in S^n. If we use this data to fit (3.53), virtually every fitted value of θ_f will be nonzero, producing a huge model with little explanatory power. To overcome this, we introduce what is known as a *regularization* term where we penalize nonzero values of θ. We would write the optimization problem as

$$\min_{\theta} \left(\sum_{n=1}^{N} (f^n - \overline{F}(S^n|\theta))^2 + \lambda \sum_{f \in \mathcal{F}} \|\theta_f\|_1 \right). \tag{3.54}$$

Here, we are introducing a penalty of $\lambda \|\theta_f\|_1$ which represents the regularization term. As we increase λ, we put a higher penalty for allowing θ_f to be in the model. It is necessary to increase λ, take the resulting model, and then test it on an out-of-sample dataset. Typically, this is done repeatedly (five times is typical) where the out-of-sample observations are drawn from a different 20 percent of the data (this process is known as *cross-validation*). We can plot the error from this testing for each value of λ, and find the best value of λ.

A variant uses the L_2 norm, which minimizes the sum of squares of the elements of θ, as in

$$\min_{\theta} \left(\sum_{n=1}^{N} (f^n - \overline{F}(S^n|\theta))^2 + \lambda \sum_{f \in \mathcal{F}} \|\theta_f\|_2 \right). \tag{3.55}$$

The L_2 norm is easier to solve, but allows small values of θ to enter the solution.

This procedure is known as Lasso, for "Least absolute shrinkage and selection operator." The procedure is inherently batch, although there is a recursive form that has been developed. The method works best when we assume there is access to an initial testing dataset that can be used to help identify the best set of features.

A challenge with regularization is that it requires determining the best value of λ. It should not be surprising that you will get the best fit if you set $\lambda = 0$, creating a model with a large number of parameters. The problem is that these models do not offer the best predictive power, because many of the fitted parameters $\theta_f > 0$ reflect spurious noise rather than the identification of truly important features.

The way to overcome this is to use cross-validation, which works as follows. Imagine fitting the model on an 80 percent sample of the data, and then evaluating the model on the remaining 20 percent. Now, repeat this five times by rotating through the dataset, using different portions of the data for testing. Finally, repeat this entire process for different values of λ to find the value of λ that produces the lowest error.

Regularization is sometimes referred to as modern statistics. While not an issue for very low dimensional models where all the variables are clearly important, regularization is arguably one of the most powerful tools for modern models which feature large numbers of variables. Regularization can be introduced into virtually any statistical model, including nonlinear models and neural networks.

3.8 RECURSIVE LEAST SQUARES FOR LINEAR MODELS

Perhaps one of the most appealing features of linear regression is the ease with which models can be updated recursively. Recursive methods are well known in the statistics and machine learning communities, but these communities often focus on batch methods. Recursive statistics is especially valuable in stochastic optimization because they are well suited to any adaptive algorithm.

We start with a basic linear model

$$y = \theta^T x + \varepsilon,$$

where $\theta = (\theta_1, \ldots, \theta_I)^T$ is a vector of regression coefficients. We let X^n be the $n \times I$ matrix of observations (where n is the number of observations). Using batch statistics, we can estimate θ from the normal equation

$$\theta = [(X^n)^T X^n]^{-1} (X^n)^T Y^n. \tag{3.56}$$

We note in passing that equation (3.56) represents an optimal solution of a statistical model using a sampled dataset, one of the major solution strategies that we are going to describe in chapter 4 (stay tuned!).

We now make the conversion to the vocabulary where instead of a feature x_i, we are going to let x be our data and let $\phi_f(x)$ be a feature (also known as basis functions), where $f \in \mathcal{F}$ is our set of features. We let $\phi(x)$ be a column vector of the features, where $\phi^n = \phi(x^n)$ replaces x^n. We also write our function approximation using

$$\overline{F}(x|\theta) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(x) = \phi(x)^T \theta.$$

Throughout our presentation, we assume that we have access to an observation $\hat{f}^n$ of our function $F(x, W)$.

3.8.1 Recursive least squares for stationary data

In the setting of adaptive algorithms in stochastic optimization, estimating the coefficient vector θ using batch methods such as equation (3.56) would be very expensive. Fortunately, it is possible to compute these formulas recursively. The updating equation for θ is

$$\theta^n = \theta^{n-1} - H^n \phi^n \hat{\varepsilon}^n, \tag{3.57}$$

where H^n is a matrix computed using

$$H^n = \frac{1}{\gamma^n} B^{n-1}. \tag{3.58}$$

The error $\hat{\varepsilon}^n$ is computed using

$$\hat{\varepsilon}^n = \overline{F}(x|\theta^{n-1}) - \hat{y}^n. \tag{3.59}$$

Note that it is common in statistics to compute the error in a regression using "actual minus predicted" while we are using "predicted minus actual" (see equation (3.59) above). Our sign convention is motivated by the derivation from first principles of optimization, which we cover in more depth in chapter 5. B^{n-1} is an $|\mathcal{F}|$ by $|\mathcal{F}|$ matrix which is updated recursively using

$$B^n = B^{n-1} - \frac{1}{\gamma^n}(B^{n-1}\phi^n(\phi^n)^T B^{n-1}). \tag{3.60}$$

γ^n is a scalar computed using

$$\gamma^n \quad = \quad 1 + (\phi^n)^T B^{n-1}\phi^n. \tag{3.61}$$

The derivation of equations (3.57)-(3.61) is given in section 3.13.1.

It is possible in any regression problem that the matrix $(X^n)^T X^n$ (in equation (3.56)) is non-invertible. If this is the case, then our recursive formulas are not going to overcome this problem. When this happens, we will observe $\gamma^n = 0$. Alternatively, the matrix may be invertible, but unstable, which occurs when γ^n is very small (say, $\gamma^n < \epsilon$ for some small ϵ). When this occurs, the problem can be circumvented by using

$$\bar{\gamma}^n = \gamma^n + \delta,$$

where δ is a suitably chosen small perturbation that is large enough to avoid instabilities. Some experimentation is likely to be necessary, since the right value depends on the scale of the parameters being estimated.

The only missing step in our algorithm is initializing B^n. One strategy is to collect a sample of m observations where m is large enough to compute B^m using full inversion. Once we have B^m, we can then proceed to update it using the formula above. A second strategy is to use $B^0 = \epsilon I$, where I is the identity matrix and ϵ is a "small constant." This strategy is not guaranteed to give the exact values, but should work well if the number of observations is relatively large.

In our stochastic optimization applications, the observations $\hat{f}^n$ will represent observations of the value of a function, or estimates of the value of being in a state, or even decisions we should make given a state. Our data can be a decision x (or possibly the decision x and initial state S_0), or a state S. The updating equations assume implicitly that the estimates come from a stationary series.

There are many problems where the number of basis functions can be extremely large. In these cases, even the efficient recursive expressions in this section cannot avoid the fact that we are still updating a matrix where the number of rows and columns may be large. If we are only estimating a few dozen or a few hundred parameters, this can be fine. If the number of parameters extends into the thousands, even this strategy would probably bog down. It is very important to work out the approximate dimensionality of the matrices before using these methods.

3.8.2 Recursive least squares for nonstationary data

It is generally the case in approximate dynamic programming that our observations $\hat{f}^n$ (typically, updates to an estimate of a value function) come from a nonstationary process.

This is true even when we are estimating the value of a fixed policy if we use TD learning, but it is always true when we introduce the dimension of optimizing over policies. Recursive least squares puts equal weight on all prior observations, whereas we would prefer to put more weight on more recent observations.

It is not uncommon that our data is coming from a nonstationary series. For example, in an online setting, our function $F(x, W)$ may be evolving over time. For example, x may be a price and $F(x, W)$ may be the number of people buying a product or service in a dynamic, evolving marketplace. Alternatively, if we are approximating the value of being in a state $V(S)$, it is not uncommon that the driving data is nonstationary (we will see this in chapters 17 and 18).

Instead of minimizing total errors (as we do in equation (3.50)) it makes sense to minimize a geometrically weighted sum of errors

$$\min_{\theta} \sum_{m=1}^{n} \lambda^{n-m} \left(f^m - (\theta_0 + \sum_{i=1}^{I} \theta_i \phi_i^m) \right)^2, \tag{3.62}$$

where λ is a discount factor that we use to discount older observations. If we repeat the derivation in section 3.8.1, the only changes we have to make are in the updating formula for B^n, which is now given by

$$B^n = \frac{1}{\lambda} \left(B^{n-1} - \frac{1}{\gamma^n} (B^{n-1} \phi^n (\phi^n)^T B^{n-1}) \right), \tag{3.63}$$

and the expression for γ^n, which is now given by

$$\gamma^n \quad = \quad \lambda + (\phi^n)^T B^{n-1} \phi^n. \tag{3.64}$$

λ works in a way similar to a stepsize, although in the opposite direction. Setting $\lambda = 1$ means we are putting an equal weight on all observations, while smaller values of λ puts more weight on more recent observations. In this way, λ plays a role similar to our use of λ in TD(λ).

We could use this logic and view λ as a tunable parameter. Of course, a constant goal in the design of algorithms is to avoid the need to tune yet another parameter. For the special case where our regression model is just a constant (in which case $\phi^n = 1$), we can develop a simple relationship between α_n and the discount factor (which we now compute at each iteration, so we write it as λ_n). Let $G^n = (H^n)^{-1}$, which means that our updating equation is now given by

$$\theta^n = \theta^{n-1} - (G^n)^{-1} \phi^n \hat{\varepsilon}^n.$$

Recall that we compute the error ε^n as predicted minus actual as given in equation (3.59). This is required if we are going to derive our optimization algorithm based on first principles, which means that we are minimizing a stochastic function. The matrix G^n is updated recursively using

$$G^n = \lambda_n G^{n-1} + \phi^n (\phi^n)^T, \tag{3.65}$$

with $G^0 = 0$. For the case where $\phi^n = 1$ (in which case G^n is also a scalar), $(G^n)^{-1} \phi^n = (G^n)^{-1}$ plays the role of our stepsize, so we would like to write $\alpha_n = G^n$. Assume that $\alpha_{n-1} = \left(G^{n-1}\right)^{-1}$. Equation (3.65) implies that

$$\begin{aligned} \alpha_n &= (\lambda_n G^{n-1} + 1)^{-1} \\ &= \left(\frac{\lambda_n}{\alpha_{n-1}} + 1 \right)^{-1}. \end{aligned}$$

Solving for λ_n gives

$$\lambda_n = \alpha_{n-1}\left(\frac{1-\alpha_n}{\alpha_n}\right). \tag{3.66}$$

Note that if $\lambda_n = 1$, then we want to put equal weight on all the observations (which would be optimal if we have stationary data). We know that in this setting, the best stepsize is $\alpha_n = 1/n$. Substituting this stepsize into equation (3.66) verifies this identity.

The value of equation (3.66) is that it allows us to relate the discounting produced by λ_n to the choice of stepsize rule, which has to be chosen to reflect the nonstationarity of the observations. In chapter 6, we introduce a much broader range of stepsize rules, some of which have tunable parameters. Using (3.66) allows us to avoid introducing yet another tunable parameter.

3.8.3 Recursive estimation using multiple observations

The previous methods assume that we get one observation and use it to update the parameters. Another strategy is to sample several paths and solve a classical least-squares problem for estimating the parameters. In the simplest implementation, we would choose a set of realizations $\hat{\Omega}^n$ (rather than a single sample ω^n) and follow all of them, producing a set of estimates $(f(\omega))_{\omega\in\hat{\Omega}^n}$ that we can use to update our estimate of the function $\overline{F}(s|\theta)$.

If we have a set of observations, we then face the classical problem of finding a vector of parameters $\hat{\theta}^n$ that best match all of these function estimates. Thus, we want to solve

$$\hat{\theta}^n = \arg\min_{\theta} \frac{1}{|\hat{\Omega}^n|} \sum_{\omega\in\hat{\Omega}^n} (\overline{F}(s|\theta) - f(\omega))^2.$$

This is the standard parameter estimation problem faced in the statistical estimation community. If $\overline{F}(s|\theta)$ is linear in θ, then we can use the usual formulas for linear regression. If the function is more general, we would typically resort to nonlinear programming algorithms to solve the problem. In either case, $\hat{\theta}^n$ is still an update that needs to be smoothed in with the previous estimate θ^{n-1}, which we would do using

$$\theta^n = (1-\alpha_{n-1})\theta^{n-1} + \alpha_{n-1}\hat{\theta}^n. \tag{3.67}$$

One advantage of this strategy is that in contrast with the updates that depend on the gradient of the value function, updates of the form given in equation (3.67) do not encounter a scaling problem, and therefore we return to our more familiar territory where $0 < \alpha_n \leq 1$. Of course, as the sample size $\hat{\Omega}$ increases, the stepsize should also be increased because there is more information in $\hat{\theta}^n$. Using stepsizes based on the Kalman filter (sections 6.3.2 and 6.3.3) will automatically adjust to the amount of noise in the estimate.

The usefulness of this particular strategy will be very problem-dependent. In many applications, the computational burden of producing multiple estimates $\hat{v}^n(\omega), \omega \in \hat{\Omega}^n$ before producing a parameter update will simply be too costly.

3.9 NONLINEAR PARAMETRIC MODELS

While linear models are exceptionally powerful (recall that "linear" means linear in the parameters), it is inevitable that some problems will require models that are nonlinear in the parameters. We might want to model the nonlinear response of price, dosage, or temperature. Nonlinear models introduce challenges in model estimation as well as learning in stochastic optimization problems.

We begin with a presentation on maximum likelihood estimation, one of the most widely used estimation methods for nonlinear models. We then introduce the idea of a sampled nonlinear model, which is a simple way of overcoming the complexity of a nonlinear model. We close with an introduction to neural networks, a powerful approximation architecture that has proven to be useful in machine learning as well as dynamic programs arising in engineering control problems.

3.9.1 Maximum likelihood estimation

The most general method for estimating nonlinear models is known as maximum likelihood estimation. Let $f(x|\theta)$ the function given θ, and assume that we observe

$$y = f(x|\theta) + \epsilon$$

where $\epsilon \sim N(0, \sigma^2)$ is the error with density

$$f^{\epsilon}(w) = \frac{1}{\sqrt{2\pi}\sigma} \exp \frac{w^2}{2\sigma^2}.$$

Now imagine that we have a set of observations $(y^n, x^n)_{n=1}^N$. The likelihood of observing $(y^n)_{n=1}^N$ is given by

$$L(y|x,\theta) = \Pi_{n=1}^N \exp \frac{(y^n - f(x^n|\theta))^2}{2\sigma^2}.$$

It is common to use the log likelihood $\mathcal{L}(y|x,\theta) = \log L(y|x,\theta)$, which gives us

$$\mathcal{L}(y|x,\theta) = \sum_{n=1}^N \frac{1}{\sqrt{2\pi}\sigma}(y^n - f(x^n|\theta))^2, \tag{3.68}$$

where we can, of course, drop the leading constant $\frac{1}{\sqrt{2\pi}\sigma}$ when maximizing $\mathcal{L}(y|x,\theta)$.

Equation (3.68) can be used by nonlinear programming algorithms to estimate the parameter vector θ. This assumes that we have a batch dataset $(y^n, x^n)_{n=1}^N$, which is not our typical setting. In addition, the log likelihood $\mathcal{L}(y|x,\theta)$ can be nonconvex when $f(x|\theta)$ is nonlinear in θ, which further complicates the optimization challenge.

The next section describes a method for handling nonlinear models in a recursive setting.

3.9.2 Sampled nonlinear models

A powerful strategy for estimating models that are nonlinear in the parameters assumes that the unknown parameter θ can only take on one of a finite set $\theta_1, \theta_2, \ldots, \theta_K$. Let $\boldsymbol{\theta}$ be a random variable representing the true value of θ, where $\boldsymbol{\theta}$ takes on one of the values $(\theta_k)_{k=1}^K$. Assume we start with a prior set of probabilities $p_k^0 = \mathbb{P}[\boldsymbol{\theta} = \theta_k]$. This is framework we use when we adopt a Bayesian perspective: we view the true value of θ as a random variable $\boldsymbol{\theta}$, with a prior distribution of belief p^0 (which might be uniform).

What we are now going to do is to use observations of the random variable Y to update our probability distribution. To illustrate this, assume that we are observing successes and failures, so $Y \in \{0,1\}$, as might happen with medical outcomes. In this setting, the vector x would consist of information about a patient as well as medical decisions. Assume that the probability that $Y = 1$ is given by a logistic regression, given by

$$\begin{aligned} f(y|x,\theta) &= \mathbb{P}[Y=1|x,\theta] &(3.69)\\ &= \frac{\exp^{U(x|\theta)}}{1+\exp^{U(x|\theta)}}, &(3.70)\end{aligned}$$

where $U(x|\theta)$ is a linear model given by

$$U(x|\theta) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_M.$$

We assume that θ is one of the elements $(\theta_k)_{k=1}^K$, where θ_k is a vector of elements $(\theta_{km})_{m=1}^M$. Let $H^n = (y^1, \ldots, y^n)$ be our history of observations of the random outcome Y. Now assume that $p_k^n = \mathbb{P}[\boldsymbol{\theta} = \theta_k|H^n]$, and that we next choose x^n and observe $Y = y^{n+1}$ (later, we are going to talk about how to choose x^n). We can update our probabilities using Bayes theorem

$$p_k^{n+1} = \frac{\mathbb{P}[Y = y^{n+1}|H^n, x^n, \theta_k]\mathbb{P}[\boldsymbol{\theta} = \theta_k]}{\mathbb{P}[Y = y^{n+1}|H^n, x^n]}. \qquad (3.71)$$

We start by observing that $\mathbb{P}[\boldsymbol{\theta} = \theta_k] = p_k^n$. The conditional probability $\mathbb{P}[Y = y^{n+1}|x^n, \theta_k]$ comes our from our logistic regression in (7.37):

$$\mathbb{P}[Y = y^{n+1}|H^n, x^n, \theta_k] = \begin{cases} f(x^n|\theta^n) & \text{If } y^{n+1} = 1, \\ 1 - f(x^n|\theta^n) & \text{If } y^{n+1} = 0. \end{cases}$$

Finally, we compute the denominator using

$$\mathbb{P}[Y = y^{n+1}|H^n, x^n] = \sum_{k=1}^K \mathbb{P}[Y = y^{n+1}|H^n, x^n, \theta_k]p_k^n.$$

This idea can be extended to a wide range of distributions for Y. Its only limitation (which may be significant) is the assumption that θ can be only one of a finite set of discrete values. A strategy for overcoming this limitation is to periodically generate new possible values of θ, use the past history of observations to obtain updated probabilities, and then drop the values with the lowest probability.

3.9.3 Neural networks - I

Neural networks represent an unusually powerful and general class of approximation strategies that have been widely used in optimal control and statistical learning. There are a number of excellent textbooks on the topic, so our presentation is designed only to introduce the basic idea and encourage readers to experiment with this technology if simpler models are not effective. In this section, we restrict our attention to low-dimensional neural networks, which we describe below.

3.9.3.1 The basic idea Up to now, we have considered approximation functions of the form

$$\overline{F}(x|\theta) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(x),$$

where $\mathcal{F}$ is our set of features, and $(\phi_f(x))_{f\in\mathcal{F}}$ are the basis functions which extract what are felt to be the important characteristics of the state variable which explain the value of being in a state. We have seen that when we use an approximation that is linear in the parameters, we can estimate the parameters θ recursively using standard methods from linear regression. For example, if R_i is the number of resources of type i, our approximation might look like

$$\overline{F}(R|\theta) = \sum_{i\in\mathcal{I}} \left(\theta_{1i}R_i + \theta_{2i}R_i^2\right).$$

Now assume that we feel that the best function might not be quadratic in R_i, but we are not sure of the precise form. We might want to estimate a function of the form

$$\overline{F}(R|\theta) = \sum_{i\in\mathcal{I}} \left(\theta_{1i}R_i + \theta_{2i}R_i^{\theta_3}\right).$$

Now we have a function that is nonlinear in the parameter vector $(\theta_1, \theta_2, \theta_3)$, where θ_1 and θ_2 are vectors and θ_3 is a scalar. If we have a training dataset of state-value observations, $(\hat{f}^n, R^n)_{n=1}^N$, we can find θ by solving

$$\min_\theta \sum_{n=1}^N \left(\hat{f}^n - \overline{F}(R^n|\theta)\right)^2,$$

which generally requires the use of nonlinear programming algorithms. One challenge is that nonlinear optimization problems do not lend themselves to the simple recursive updating equations that we obtained for linear (in the parameters) functions. But more problematic is that we have to experiment with various functional forms to find the one that fits best.

Neural networks offer a much more flexible set of architectures, and at the same time can be updated recursively. The technology has matured to the point that there are a number of commercial packages available which implement the algorithms. However, applying the technology to specific dynamic programming problems can be a nontrivial challenge. In addition, it is not possible to know in advance which problem classes will benefit most from the additional generality in contrast with the simpler strategies that we have covered in this chapter.

Neural networks are, ultimately, a form of statistical model which can be used to approximate the function $\mathbb{E}f(x,W)$ (or a policy $X^\pi(S)$, or a value function $V(S)$). We will have an input x (or S), and we are using a neural network to predict an output $\hat{f}$ (or a decision x^n, or a value f^n). Using the traditional notation of statistics, let x^n be a vector of inputs which could be features $\phi_f(S^n)$ for $f \in \mathcal{F}$. If we were using a linear model, we would write

$$f(x^n|\theta) = \theta_0 + \sum_{i=1}^I \theta_i x_i^n.$$

In the language of neural networks, we have I inputs (we have $I+1$ parameters since we also include a constant term), which we wish to use to estimate a single output f^{n+1} (a random observations of our function). The relationships are illustrated in figure 3.6 where we show the I inputs which are then "flowed" along the links to produce $f(x^n|\theta)$. After this, we then learn the sample realization $\hat{f}^{n+1}$ that we were trying to predict, which allows

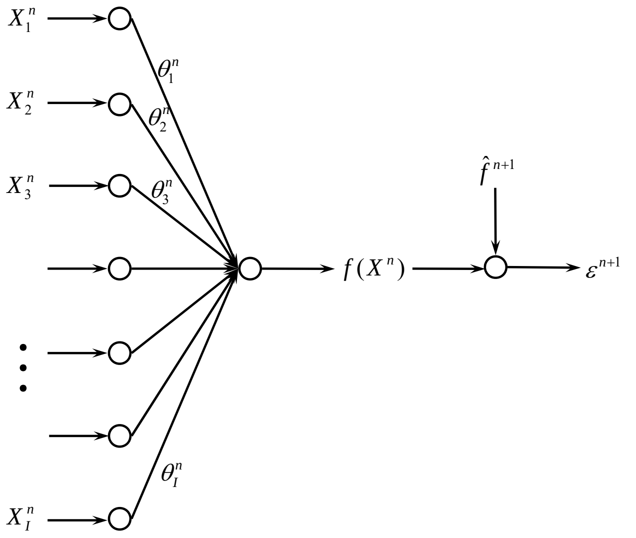


Figure 3.6 Neural networks with a single layer.

us to compute the error $\epsilon^{n+1} = \hat{f}^{n+1} - f(x^n|\theta)$. Define the random variable X to describe a set of inputs (where x^n is the value of X at the nth iteration), and let $\hat{f}$ be the random variable giving the response from input X. We would like to find a vector θ that solves

$$\min_\theta \mathbb{E}\frac{1}{2}(f(X|\theta) - \hat{f})^2.$$

Let $F(\theta) = \mathbb{E}\big(0.5(f(X|\theta) - \hat{f})^2\big)$, and let $F(\theta, \hat{f}) = 0.5(f(X|\theta) - \hat{f})^2$ where $\hat{f}$ is a sample realization of our function. As before, we can solve this iteratively using the algorithm we first introduced in section 3.2 which gives us the updating equation

$$\theta^{n+1} = \theta^n - \alpha_n \nabla_\theta F(\theta^n, \hat{f}^{n+1}),$$

where $\nabla_\theta F(\theta^n, \hat{f}^{n+1}) = \epsilon^{n+1}$.

We illustrated our linear model by assuming that the inputs were the individual dimensions of the control variable which we denoted X_i^n. We may not feel that this is the best way to represent the state of the system (imagine representing the states of a Connect-4 game board). We may feel it is more effective (and certainly more compact) if we have access to a set of basis functions $\phi_f(X),\ f \in \mathcal{F}$, where $\phi_f(X)$ captures a relevant feature of our system given the inputs X. In this case, we would be using our standard basis function representation, where each basis function provides one of the inputs to our neural network.

This was a simple illustration, but it shows that if we have a linear model, we get the same basic class of algorithms that we have already used. A richer model, given in figure 3.7, illustrates a more classical neural network. Here, the "input signal" X^n (this can be the state variable or the set of basis functions) is communicated through several layers. Let $x^{(1,n)} = X^n$ be the input to the first layer (recall that X_i^n might be the i^{th} dimension of the state variable itself, or a basis function). Let $\mathcal{I}^{(1)}$ be the set of inputs to the first layer (for example, the set of basis functions).

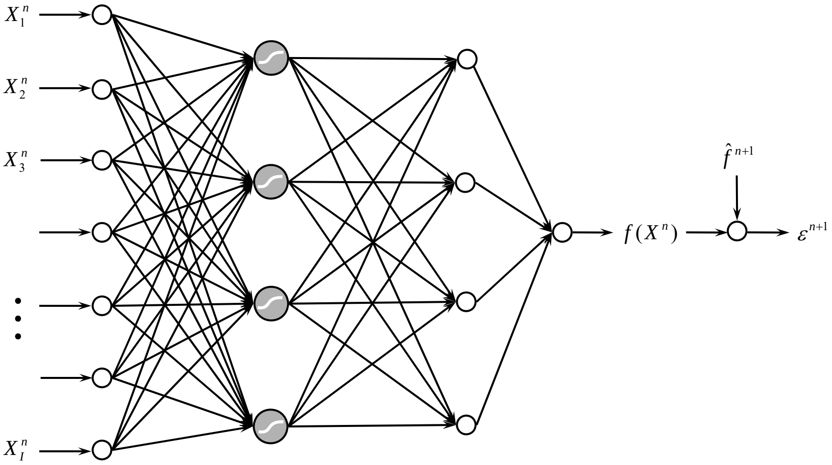


Figure 3.7 A three-layer neural network.

Here, the first linear layer produces J outputs given by

$$Y_j^{(2,n)} = \sum_{i \in \mathcal{I}^{(1)}} \theta_{ij}^{(1)} X_i^{(1,n)}, \qquad j \in \mathcal{I}^{(2)}.$$

$X_j^{(2,n)}$ becomes the input to a nonlinear *perceptron* node which is characterized by a nonlinear function that may dampen or magnify the input. A typical functional form for a perceptron node is the logistics function given by

$$\sigma(y) = \frac{1}{1 + e^{-\beta y}},$$

where β is a scaling coefficient. The function $\sigma(y)$ is illustrated in figure 3.8. The sigmoid function $\sigma(x)$ introduces nonlinear behavior into the communication of the "signal" X^n.

We next calculate

$$X_i^{(2,n)} = \sigma(Y_i^{(2,n)}), \;\; i \in \mathcal{I}^{(2)}$$

and use $X_i^{(2,n)}$ as the input to the second linear layer. We then compute

$$X_j^{(3,n)} = \sum_{i \in \mathcal{I}^{(2)}} \theta_{ij}^{(2)} X_i^{(2,n)}, \qquad j \in \mathcal{I}^{(3)}.$$

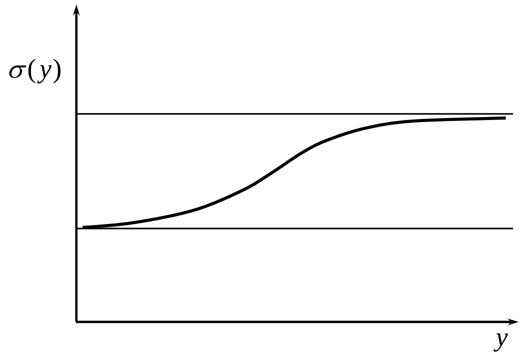


Figure 3.8 Illustrative logistics function for introducing nonlinear behavior into neural networks.

Finally, we compute the single output using

$$f^n = \sum_{i \in \mathcal{I}^{(3)}} \theta_i^{(3)} X_i^{(3,n)}.$$

As before, f^n is our estimate of the response from input X^n. This is our function approximation $\overline{F}^n(s|\theta)$ which we update using the observation f^{n+1}. We update the parameter vector $\theta = (\theta^{(1)}, \theta^{(2)}, \theta^{(3)})$ using the same stochastic gradient algorithms we used for a single layer network. The only difference is that the derivatives have to capture the fact that changing $\theta^{(1)}$, for example, impacts the "flows" through the rest of the network. The derivatives are slightly more difficult to compute, but the logic is basically the same.

Our presentation above assumes that there is a single output, which is to say that we are trying to match a scalar quantity $\hat{f}^{n+1}$, the observed value of a function. In some settings, $\hat{f}^{n+1}$ might be a vector. For example, in chapter 19 (see, in particular, section 19.4), we describe problems where $\hat{f}^{n+1}$ is the gradient of a value function, which of course would be multidimensional. In this case, f^n would also be a vector which would be estimated using

$$f_j^n = \sum_{i \in \mathcal{I}^{(3)}} \theta_{ij}^{(3)} X_i^{(3,n)}, \quad j \in \mathcal{I}^{(4)},$$

where $|\mathcal{I}^{(4)}|$ is the dimensionality of the output layer (that is, the dimensionality of $\hat{f}^{n+1}$).

This presentation should be viewed as nothing more than a very simple illustration of an extremely rich field. The advantage of neural networks is that they offer a much richer class of nonlinear functions ("nonlinear architectures" in the language of neural networks) which can be trained in an iterative way. Calculations involving neural networks exploit the layered structure, and naturally come in two forms: feed forward propagation, where we step forward through the layers "simulating" the evolution of the input variables to the outputs, and backpropagation, which is used to compute derivatives so we can calculate the marginal impact of changes in the parameters. We describe these two processes below.

3.9.3.2 Feed Forward Propagation Between every two layers the inputs from the previous layer are, first, linearly (through the weight matrices) propagated forward and then nonlinearly transformed (through the sigmoids) to produce an output on the successor layer. Recursing this process, which we refer to as feed forward propagation, over the total layers of the network will produce an output on the final layer L. In what comes next, we will present an efficient vectorized implementation of forward propagation.

We call the computation performed by node i in layer l an activation and denote it by $a_i^{(l)} = \text{sigmoid}\left(\sum_{j=0}^{s_{l-1}} \boldsymbol{\theta}_{ij}^{(l-1)} a_j^{(l-1)}\right)$, with $a_0^{(l-1)} = 1$, and $a_j^{(l-1)}$ for $j \in \{1, \ldots, s_{l-1}\}$ being the activations of the previous layer provided that the activations of the input (or first layer) are the data points themselves (that is, the different dimensions of the state variable). Clearly, $a_i^{(l)}$ can be written in a matrix-vector product form with the help of $\boldsymbol{\theta}^{(l-1)}$. Essentially, the input to the sigmoidal function is a linear combination between the i^{th} row of the weight matrix and the activations on the previous layer:

$$a_i^{(l)} = \text{sigmoid}\left(\boldsymbol{\theta}_{i,:}^{(l-1),\mathsf{T}} \boldsymbol{a}^{(l-1)}\right) = \frac{1}{1 + e^{-\boldsymbol{\theta}_{i,:}^{(l-1),\mathsf{T}} \boldsymbol{a}^{(l-1)}}},$$

where $\boldsymbol{\theta}_{i,:}^{(l-1),\mathsf{T}}$ denotes the i^{th} row of $\boldsymbol{\theta}^{(l-1)}$, and $\boldsymbol{a}^{(l-1)}$ is a vector collecting all activations (including that of the bias term) of layer $l-1$, i.e., $\boldsymbol{a}^{(l-1)} = \left[1, a_1^{(l-1)}, a_2^{(l-1)}, \ldots, a_{s_{l-1}}^{(l-1)}\right]^{\mathsf{T}}$. Now, we can easily generalize the above notion to compute the activations in layer l by considering the whole matrix $\boldsymbol{\theta}^{(l-1)}$:

$$\begin{aligned}
\boldsymbol{a}^{(l)} &= \text{sigmoid}\left(\boldsymbol{\theta}^{(l-1)} \boldsymbol{a}^{(l-1)}\right) \\
&= \text{sigmoid}\left[\begin{pmatrix} \boldsymbol{\theta}_{10}^{(l-1)} & \boldsymbol{\theta}_{12}^{(l-1)} & \cdots & \boldsymbol{\theta}_{1s_{l-1}}^{(l-1)} \\ \boldsymbol{\theta}_{20}^{(l-1)} & \boldsymbol{\theta}_{21}^{(l-1)} & \cdots & \boldsymbol{\theta}_{2s_{l-1}}^{(l-1)} \\ \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{\theta}_{s_{l+1}0}^{(l-1)} & \boldsymbol{\theta}_{s_{l+1}1}^{(l-1)} & \cdots & \boldsymbol{\theta}_{s_{l+1}s_{l-1}}^{(l-1)} \end{pmatrix} \begin{bmatrix} 1 \\ a_1^{(l-1)} \\ \vdots \\ a_{s_{l-1}}^{(l-1)} \end{bmatrix}\right] \\
&= \begin{bmatrix} \text{sigmoid}\left(\boldsymbol{\theta}_{1,:}^{(l-1),\mathsf{T}} \boldsymbol{a}^{(l-1)}\right) \\ \text{sigmoid}\left(\boldsymbol{\theta}_{2,:}^{(l-1),\mathsf{T}} \boldsymbol{a}^{(l-1)}\right) \\ \vdots \\ \text{sigmoid}\left(\boldsymbol{\theta}_{s_{l+1},:}^{(l-1),\mathsf{T}} \boldsymbol{a}^{(l-1)}\right) \end{bmatrix}.
\end{aligned}$$

3.9.3.3 Backpropagation Having described feed forward propagation, the next step is to detail the strategy by which neural networks determine the model parameters (i.e., the $\boldsymbol{\theta}^{(l)}$ matrices). In standard regression or classification problems (which is to say, supervised learning), back-propagation is the most common algorithmic approach. Given an input data point, back-propagation commences as follows. First, forward propagation is executed and the network is made to output a value. This value is then compared to the real output from the data set producing an error. This error is then propagated backwards to every other layer and used to update connecting weights. Such updates typically involve gradient-based methods (e.g., stochastic gradients).

When attempting to apply back-propagation for neural networks in dynamic programming, one is faced with the problem of determining a good performance measure since we are trying to produce controls without knowing what the true control should be (this is what we would have if we were using the neural network as a statistical model to fit data). Instead, we are going to tune the weights θ to minimize cost/maximize contribution.

3.9.4 Comments

It is important to emphasize that neural networks are no panacea (a statement that can be made about almost anything). As with our simple linear models, the updating mechanisms struggle with scaling (the units of X and the units of f^{n+1} may be completely different) which has to be handled by the parameter vectors $\theta^{(\ell)}$. Neural networks are high-dimensional models, which means they can fit almost anything. The price of this flexibility is that they require very large datasets. This is a reason they have been so successful at pattern matching applications such as voice recognition or identifying faces or classifying news articles. Neural networks can be trained with vast databases.

3.10 NONPARAMETRIC MODELS

The power of parametric models is matched by their fundamental weakness: they are only effective if you can design an effective parametric model, and this remains a frustrating art. For this reason, nonparametric statistics have attracted recent attention. They avoid the art of specifying a parametric model, but introduce other complications. Nonparametric methods work primarily by building local approximations to functions using observations rather than depending on functional approximations.

Nonparametric models are characterized by the property that as the number of observations N grows, we can approximate any function with arbitrary accuracy. While this sounds like a fantastic property, it comes with the price that these functions need arbitrarily large numbers of parameters (in some cases, the entire dataset).

There is an extensive literature on the use of approximation methods for continuous functions. These problems, which arise in many applications in engineering and economics, require the use of approximation methods that can adapt to a wide range of functions. Interpolation techniques, orthogonal polynomials, Fourier approximations and splines are just some of the most popular techniques. Often, these methods are used to closely approximate the expectation using a variety of numerical approximation techniques.

We note that lookup tables are, technically, a form of nonparametric approximation methods, although these can also be expressed as parametric models by using indicator variables. For example, assume that $\mathcal{X} = \{x_1, x_2, \ldots, x_M\}$ is a set of discrete inputs, and let

$$\mathbb{1}_{\{X=x\}} = \begin{cases} 1 & \text{If } X = x \in \mathcal{X}, \\ 0 & \text{Otherwise.} \end{cases}$$

be an indicator variable that tells us when X takes on a particular value. We can write our function as

$$f(X|\theta) = \sum_{x \in \mathcal{X}} \theta_x \mathbb{1}_{\{X=x\}}$$

This means that we need to estimate a parameter θ_x for each $x \in \mathcal{X}$. In principle, this is a parametric representation, but the parameter vector θ has the same dimensionality as the input vector x. However, the working definition of a nonparametric model is one that, given an infinite dataset, will produce a perfect representation of the true function, a property that our lookup table model clearly satisfies. It is precisely for this reason that we treat lookup tables as a special case since parametric models are always used for settings where the parameter vector θ is much lower dimensional than the size of $\mathcal{X}$.

In this section, we review some of the nonparametric methods that have received the most attention within the approximate dynamic programming community. This is an active area of research which offers potential as an approximation strategy, but significant hurdles remain before this approach can be widely adopted. We start with the simplest methods, closing with a powerful class of nonparametric methods known as support vector machines.

3.10.1 K-nearest neighbor

Perhaps the simplest form of nonparametric regression forms estimates of functions by using a weighted average of the k-nearest neighbors. As above, we assume we have a response y^n corresponding to a measurement $x^n = (x_1^n, x_2^n, \ldots, x_I^n)$. Let $\rho(x, x^n)$ be a distance metric between a query point x (in dynamic programming, this would be a state), and an observation x^n. Then let $\mathcal{N}^n(x)$ be the set of the k-nearest points to the query point x, where clearly we require $k \leq n$. Finally let $\bar{Y}^n(x)$ be the response function, which is our best estimate of the true function $Y(x)$ given the observations $x^1, \ldots, x^n$. When we use a k-nearest neighbor model, this is given by

$$\bar{Y}^n(x) = \frac{1}{k} \sum_{n \in \mathcal{N}^n(x)} y^n. \tag{3.72}$$

Thus, our best estimate of the function $Y(x)$ is made by averaging the k points nearest to the query point x.

Using a k-nearest neighbor model requires, of course, choosing k. Not surprisingly, we obtain a perfect fit of the data by using $k = 1$ if we base our error on the training dataset.

A weakness of this logic is that the estimate $\bar{Y}^n(x)$ can change abruptly as x changes continuously, as the set of nearest neighbors changes. An effective way of avoiding this behavior is using kernel regression, which uses a weighted sum of all data points.

3.10.2 Kernel regression

Kernel regression has attracted considerable attention in the statistical learning literature. As with k-nearest neighbor, kernel regression forms an estimate $\bar{Y}(x)$ by using a weighted sum of prior observations which we can write generally as

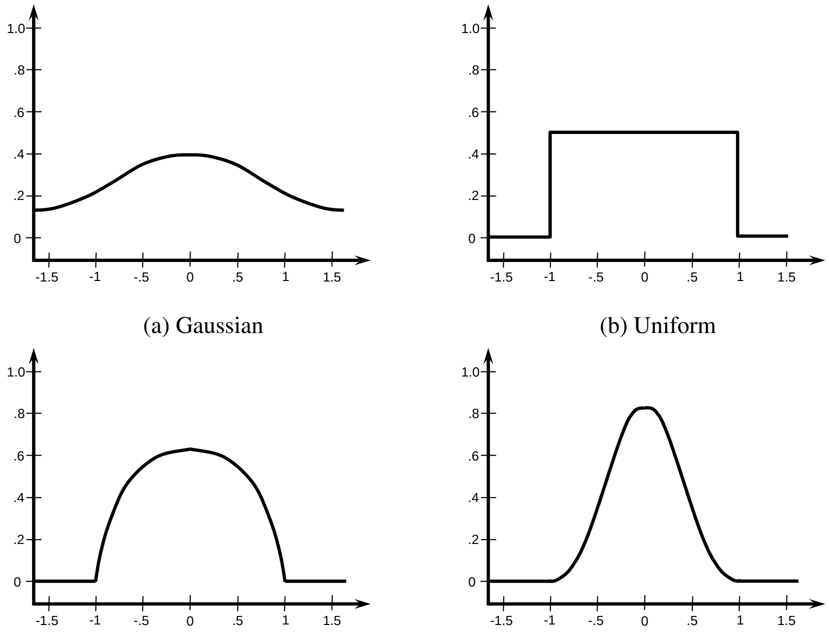
$$\bar{Y}^n(x) = \frac{\sum_{m=1}^n K_h(x, x^m) y^m}{\sum_{m=1}^n K_h(x, x^m)} \tag{3.73}$$

where $K_h(x, x^m)$ is a weighting function that declines with the distance between the query point x and the measurement x^m. h is referred to as the *bandwidth* which plays an important scaling role. There are many possible choices for the weighting function $K_h(x, x^m)$. One of the most popular is the Gaussian kernel, given by

$$K_h(x, x^m) = e^{-\left(\frac{\|x - x^m\|}{h}\right)^2}.$$

where $\|\cdot\|$ is the Euclidean norm. Here, h plays the role of the standard deviation. Note that the bandwidth h is a tunable parameter that captures the range of influence of a measurement x^m. The Gaussian kernel, often referred to as *radial basis functions* in the ADP literature, provide a smooth, continuous estimate $\bar{Y}^n(x)$. Another popular choice of kernel function is the symmetric Beta family, given by

$$K_h(x, x^m) = \max(0, (1 - \|x - x^m\|)^2)^h.$$



(a) Gaussian

(b) Uniform

(c) Epanechnikov

(d) Biweight

Figure 3.9 Illustration of Gaussian, uniform, Epanechnikov and biweight kernel weighting functions.

Here, h is a nonnegative integer. $h = 1$ gives the uniform kernel; $h = 2$ gives the Epanechnikov kernel; and $h = 3$ gives the biweight kernel. Figure 3.9 illustrates each of these four kernel functions.

We pause to briefly discuss some issues surrounding k-nearest neighbors and kernel regression. First, it is fairly common in the ADP literature to see k-nearest neighbors and kernel regression being treated as a form of aggregation. The process of giving a set of states that are aggregated together has a certain commonality with k-nearest neighbor and kernel regression, where points near each other will produce estimates of $Y(x)$ that are similar. But this is where the resemblance ends. Simple aggregation is actually a form of parametric regression using dummy variables, and it offers neither the continuous approximations, nor the asymptotic unbiasedness of kernel regression.

Kernel regression is a method of approximation that is fundamentally different from linear regression and other parametric models. Parametric models use an explicit estimation step, where each observation results in an update to a vector of parameters. At any point in time, our approximation consists of the pre-specified parametric model, along with the current estimates of the regression parameters. With kernel regression, all we do is store data until we need an estimate of the function at a query point. Only then do we trigger the approximation method, which requires looping over all previous observation, a step that clearly can become expensive as the number of observations grow.

Kernel regression enjoys an important property from an old result known as Mercer's theorem. The result states that there exists a set of basis functions $\phi_f(S),\ f \in \mathcal{F}$, possibly of very high dimensionality, where

$$K_h(S, S') = \phi(S)^T \phi(S'),$$

as long as the kernel function $K_h(S, S')$ satisfies some basic properties (satisfied by the kernels listed above). In effect this means that using appropriately designed kernels is equivalent to finding potentially very high dimensional basis functions, without having to actually create them.

Unfortunately, the news is not all good. First, there is the annoying dimension of bandwidth selection, although this can be mediated partially by scaling the explanatory variables. More seriously, kernel regression (and this includes k-nearest neighbors), cannot be immediately applied to problems with more than about five dimensions (and even this can be a stretch). The problem is that these methods are basically trying to aggregate points in a multidimensional space. As the number of dimensions grows, the density of points in the d-dimensional space becomes quite sparse, making it very difficult to use "nearby" points to form an estimate of the function. A strategy for high-dimensional applications is to use separable approximations. These methods have received considerable attention in the broader machine learning community, but have not been widely tested in an ADP setting.

3.10.3 Local polynomial regression

Classical kernel regression uses a weighted sum of responses y^n to form an estimate of $Y(x)$. An obvious generalization is to estimate locally linear regression models around each point x^n by solving a least squares problem which minimizes a weighted sum of least squares. Let $\bar{Y}^n(x|x^i)$ be a linear model around the point x^k, formed by minimizing the weighted sum of squares given by

$$\min_\theta \left(\sum_{m=1}^{n} K_h(x^k, x^m) \left(y^m - \sum_{i=1}^{I} \theta_i x_i^m \right)^2 \right). \tag{3.74}$$

Thus, we are solving a classical linear regression problem, but we do this for each point x^k, and we fit the regression using all the other points $(y^m, x^m),\ m = 1, \ldots, n$. However, we weight deviations between the fitted model and each observation y^m by the kernel weighting factor $K_h(x^k, x^m)$ which is centered on the point x^k.

Local polynomial regression offers significant advantages in modeling accuracy, but with a significant increase in complexity.

3.10.4 Neural networks - II

Low-dimensional (basically finite) neural networks are a form of parametric regression. Once you have specified the number of layers and the nodes per layer, all that is left are the weights in the network, which represent the parameters. However, there is a class of high-dimensional neural networks known as deep learners. These behave as if they have an unlimited number of layers and nodes per layer.

Deep learners have shown tremendous power in terms of their ability to capture complex patterns in language and images. It is well known that they require notoriously large datasets for training, but there are settings where there is massive amounts of data available such as the results of internet searches, images of people and text searches.

As of this writing, it is not yet clear if deep learners will prove useful in stochastic optimization, partly because our data comes from the iterations of an algorithm, and partly because the high-dimensional capabilities of neural networks raise the risk of overfitting.

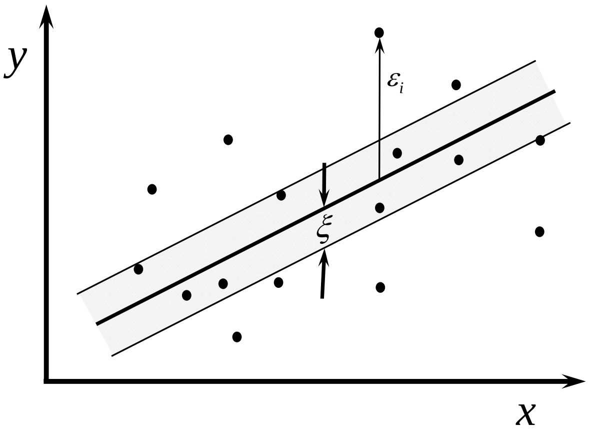


Figure 3.10 Illustration of penalty structure for support vector regression. Deviations within the gray area are assessed a value of zero. Deviations outside the gray area are measured based on their distance to the gray area.

3.10.5 Support vector machines

Support vector machines (for classification) and support vector regression (for continuous problems) have attracted considerable interest in the machine learning community. For the purpose of fitting value function approximations, we are primarily interested in support vector regression, but we can also use regression to fit policy function approximations, and if we have discrete actions, we may be interested in classification. For the moment, we focus on fitting continuous functions.

Support vector regression, in its most basic form, is linear regression with a different objective than simply minimizing the sum of the squares of the errors. With support vector regression, we consider two goals. First, we wish to minimize the absolute sum of deviations that are larger than a set amount ξ. Second, we wish to minimize the regression parameters themselves, to push as many as possible close to zero.

As before, we let our predictive model be given by

$$y = \theta x + \epsilon.$$

Let $\epsilon^i = y^i - \theta x^i$ be the error. We then choose θ by solving the following optimization problem

$$\min_{\theta} \left(\frac{\eta}{2} \|\theta\|^2 + \sum_{i=1}^{n} \max\{0, |\epsilon^i| - \xi\} \right). \tag{3.75}$$

The first term penalizes positive values of θ, encouraging the model to minimize values of θ unless they contribute in a significant way to producing a better model. The second term penalizes errors that are greater than ξ. The parameters η and ξ are both tunable parameters. The error ϵ^i and error margin ξ are illustrated in figure 3.10.

It can be shown by solving the dual that the optimal value of θ and the best fit $\bar{Y}(x)$ have the form

$$\begin{aligned}\theta &= \sum_{i=1}^{n}(\bar{\beta}^i - \bar{\alpha}^i)x^i,\\ \bar{Y}(x) &= \sum_{i=1}^{n}(\bar{\beta}^i - \bar{\alpha}^i)(x^i)^T x^i.\end{aligned}$$

Here, $\bar{\beta}^i$ and $\bar{\alpha}^i$ are scalars found by solving

$$\min_{\bar{\beta}^i,\bar{\alpha}^i} \xi \sum_{i=1}^{n}(\bar{\beta}^i + \bar{\alpha}^i) - \sum_{i=1}^{n} y^i(\bar{\beta}^i + \bar{\alpha}^i) + \frac{1}{2}\sum_{i=1}^{n}\sum_{i'=1}^{n}(\bar{\beta}^i + \bar{\alpha}^i)(\bar{\beta}^{i'} + \bar{\alpha}^{i'})(x^i)^T x^{i'},$$

subject to the constraints

$$\begin{aligned}0 \le \bar{\alpha}^i, \bar{\beta}^i &\le 1/\eta,\\ \sum_{i=1}^{n}(\bar{\beta}^i - \bar{\alpha}^i) &= 0,\\ \bar{\alpha}^i\bar{\beta}^i &= 0.\end{aligned}$$

3.10.6 Indexed functions, tree structures and clustering

There are many problems where we feel comfortable specifying a simple set of basis functions for some of the parameters, but we do not have a good feel for the nature of the contribution of other parameters. For example, we may wish to plan how much energy to hold in storage over the course of the day. Let R_t be the amount of energy stored at time t, and let H_t be the hour of the day. Our state variable might be $S_t = (R_t, H_t)$. We feel that the value of energy in storage is a concave function in R_t, but this value depends in a complex way on the hour of day. It would not make sense, for example, to specify a value function approximation using

$$\overline{V}(S_t) = \theta_0 + \theta_1 R_t + \theta_2 R_t^2 + \theta_3 H_t + \theta_4 H_t^2.$$

There is no reason to believe that the hour of day will be related to the value of energy storage in any convenient way. Instead, we can estimate a function $\overline{V}(S_t|H_t)$ given by

$$\overline{V}(S_t|h) = \theta_0(h) + \theta_1(h)R_t + \theta_2(h)R_t^2.$$

What we are doing here is estimating a linear regression model for each value of $h = H_t$. This is simply a form of lookup table using regression given a particular value of the complex variables. Imagine that we can divide our state variable S_t into two sets: the first set, f_t, contains variables where we feel comfortable capturing the relationship using linear regression. The second set, g_t, includes more complex variables whose contribution is not as easily approximated. If g_t is a discrete scalar (such as hour of day), we can consider estimating a regression model for each value of g_t. However, if g_t is a vector (possibly with continuous dimensions), then there will be too possible values.

When the vector g_t cannot be enumerated, we can resort to various clustering strategies. These fall under names such as regression trees and local polynomial regression (a form of kernel regression). These methods cluster g_t (or possibly the entire state S_t) and then fit simple regression models over subsets of data. In this case, we would create a set of clusters $\mathcal{C}^n$ based on n observations of states and values. We then fit a regression function $\overline{V}(S_t|c)$ for each cluster $c \in \mathcal{C}^n$. In traditional batch statistics, this process proceeds in two stages: clustering and then fitting. In approximate dynamic programming, we have to deal with the fact that we may change our clusters as we collect additional data.

A much more sophisticated strategy is based on a concept known as Dirichlet process mixtures. This is a fairly sophisticated technique, but the essential idea is that you form clusters that produce good fits around local polynomial regressions. However, unlike traditional cluster-then-fit methods, the idea with Dirichlet process mixtures is that membership in a cluster is probabilistic, where the probabilities depend on the query point (e.g., the state whose value we are trying to estimate).

3.11 NONSTATIONARY LEARNING

There are a number of settings where the true mean varies over time. We begin with the simplest setting where the mean may evolve up or down, but on average stays the same. We then consider the situation where the signal is steadily improving up to some unknown limit.

In chapter 7 we are going to use this in the context of optimizing functions of nonstationary random variables, or time-dependent functions of (typically) stationary random variables.

3.11.1 Nonstationary learning I - Martingale behavior

In the stationary case, we might write observations as

$$W_{t+1} = \mu + \varepsilon_{t+1},$$

where $\varepsilon \sim N(0, \sigma_\varepsilon^2)$. This means that $\mathbb{E}W_{t+1} = \mu$, which is an unchanging truth that we are trying to learn. We refer to this as the stationary case because the distribution of W_t does not depend on time.

Now assume that the true mean μ is also changing over time. We write the dynamics of the mean using

$$\mu_{t+1} = \mu_t + \delta_{t+1},$$

where δ is a random variable with distribution $N(0, \sigma_\delta^2)$. This means that $\mathbb{E}\{\mu_{t+1}|\mu_t\} = \mu_t$, which is the definition of a martingale. This means that on average, the true mean μ_{t+1} at time $t+1$ will be the same as at time t, although the actual may be different. Our observations are then made from

$$W_{t+1} = \mu_{t+1} + \varepsilon_{t+1}.$$

Typically, the variability of the mean process $\mu_1, \mu_t, \ldots, \mu_t, \ldots$ is much lower than the variance of the noise of an observation W of μ.

Now assume that μ_t is a vector with element μ_{tx}, where x will allow us to capture the performance of different drugs, paths through a network, people doing a job, or the price of a product. Let $\bar{\mu}_{tx}$ be the estimate of μ_{tx} at time t. Let Σ_t be the covariance matrix at time t, with element $\Sigma_{txx'} = Cov^n(\mu_{tx}, \mu_{tx'})$. This means we can write the distribution of μ_t as

$$\mu_t \sim N(\bar{\mu}_t, \Sigma_t).$$

This is the posterior distribution of μ^n, which is to say the distribution of μ_t given our n observations, and our prior $N(\bar{\mu}_0, \sigma_0)$. Let Σ^δ be the covariance matrix for the random vector δ describing the evolution of μ. The *predictive distribution* is the distribution of μ_{t+1} given μ_t, which we write as

$$\mu_{t+1}|\mu_t \sim N(\bar{\mu}_t, \tilde{\Sigma}_t),$$

where

$$\tilde{\Sigma}_t = \Sigma_t + \Sigma^\delta.$$

Let e_{t+1} be the error in a vector of observations W_{t+1} given by

$$e_{t+1} = W_{t+1} - \bar{\mu}_t.$$

Assume the errors are independent, let Σ^ε be the covariance matrix for e_{t+1}. The updated mean and covariance is computed using

$$\begin{aligned}\bar{\mu}_{t+1} &= \bar{\mu}_t + \tilde{\Sigma}_t\big(\Sigma^\varepsilon + \tilde{\Sigma}_t\big)^{-1} e_{t+1},\\ \Sigma_{t+1} &= \tilde{\Sigma}_t - \tilde{\Sigma}_t\big(\Sigma^\varepsilon + \tilde{\Sigma}_t\big)\tilde{\Sigma}_t.\end{aligned}$$

3.11.2 Nonstationary learning II - Transient behavior

A more general, but slightly more complex model, allows for predictable changes in θ_t. For example, we may know that θ_t is growing over time (perhaps θ_t is related to age or the population size), or we may be modeling variations in solar energy and have to capture the rising and setting of the sun.

We assume that μ_t is a vector with element x. Now assume we have a diagonal matrix M_t with factors that govern the predictable change in μ_t, allowing us to write the evolution of μ_t as

$$\mu_{t+1} = M_t\mu_t + \delta_{t+1}.$$

The evolution of the covariance matrix Σ_t becomes

$$\tilde{\Sigma}_t = M_t\Sigma_t M_t + \Sigma^\delta.$$

Now the evolution of the estimates of the mean and covariance matrix $\bar{\mu}_t$ and Σ_t are given by

$$\begin{aligned}\bar{\mu}_{t+1} &= M_t\bar{\mu}_t + \tilde{\Sigma}_t\big(\Sigma^\varepsilon + \tilde{\Sigma}_t\big)^{-1} e_{t+1},\\ \Sigma_{t+1} &= \tilde{\Sigma}_t - \tilde{\Sigma}_t\big(\Sigma^\varepsilon + \tilde{\Sigma}_t\big)\tilde{\Sigma}_t.\end{aligned}$$

Note there is no change in the formula for Σ_{t+1} since M_t is built into $\tilde{\Sigma}_t$.

3.11.3 Learning processes

There are many settings where we know that a process is improving over time up to an unknown limit. We refer to these as *learning processes* since we are modeling a process that learns as it progresses. Examples of learning processes are:

■ **EXAMPLE 3.1**

We have to choose a new basketball player x and then watch him improve as he gains playing time.

■ **EXAMPLE 3.2**

We observe the reduction in blood sugar due to diabetes medication x for a patient who has to adapt to the drug.

■ **EXAMPLE 3.3**

We are testing an algorithm where x are the parameters of the algorithm. The algorithm may be quite slow, so we have to project how good the final solution will be.

We model our process by assuming that observations come from

$$W_x^n = \mu_x^n + \varepsilon^n. \tag{3.76}$$

where the true mean μ_x^n rises according to.

$$\mu_x^n(\theta) = \theta_x^s + [\theta_x^\ell - \theta_x^s][1 - e^{-n\theta_x^r}]. \tag{3.77}$$

Here, θ_x^s is the expected starting point at $n = 0$, while θ_x^ℓ is the limiting value as $n \to \infty$. The parameter θ_x^r controls the rate at which the mean approaches θ_x^ℓ. Let $\theta = (\theta^s, \theta^\ell, \theta^r)$ be the vector of unknown parameters.

If we fix θ^r, then $\mu_x^n(\theta)$ is linear in θ^s and θ^ℓ, allowing us to use our equations for recursive least squares for linear models that we presented in section 3.8. This will produce estimates $\bar{\theta}^{s,n}(\theta^r)$ and $\bar{\theta}^{\ell,n}(\theta^r)$ for each possible value of θ^r.

To handle the one nonlinear parameter θ^r, assume that we discretize this parameter into the values $\theta_1^r, \ldots, \theta_K^r$. Let $p_k^{r,n}$ be the probability that $\theta^r = \theta_k^r$, which can be shown to be given by

$$p_k^{r,n} = \frac{L_k^n}{\sum_{k'=1}^K L_{k'}^n}$$

where L_k^n is the likelihood that $\theta^r = \theta_k^r$ which is given by

$$L_k^n \propto e^{-\left(\frac{W^{n+1} - \bar{\mu}_x^n}{\sigma_\varepsilon}\right)^2},$$

where σ_ε^2 is the variance of ε. This now allows us to write

$$\bar{\mu}_x^n(\theta) = \sum_{k=1}^K p_k^{r,n} \bar{\mu}_x^n(\theta|\theta^r).$$

This approach provides us with conditional point estimates and variances of $\bar{\theta}^{s,n}(\theta^r), \bar{\theta}^{\ell,n}(\theta^r)$ for each θ^r, along with the distribution $p^{r,n}$ for θ^r.

3.12 CHOOSING FUNCTIONAL APPROXIMATIONS

The choice of how to approximate any of the functions introduced at the beginning of this chapter remains an art form guided by science. We close our presentation with a discussion of the infamous "curse of dimensionality" followed by some thoughts on the design of effective function approximations.

3.12.1 The curse of dimensionality

There are many applications where state variables have multiple, possibly continuous dimensions. In some applications, the number of dimensions can number in the thousands.

■ **EXAMPLE 3.1**

An unmanned aerial vehicle may be described by location (three dimensions), velocity (three dimensions) and acceleration (three dimensions), in addition to fuel level. All 13 dimensions are continuous.

■ **EXAMPLE 3.2**

A utility is trying to plan the amount of energy that should be put in storage as a function of the wind history (six hourly measurements), the history of electricity spot prices (six measurements), and the demand history (six measurements).

■ **EXAMPLE 3.3**

A trader is designing a policy for selling an asset that is priced against a basket of 20 securities, creating a 20-dimensional state variable.

■ **EXAMPLE 3.4**

A car rental company has to manage its inventory of 12 different types of cars spread among 500 car rental locations, creating an inventory vector with 6,000 dimensions.

■ **EXAMPLE 3.5**

A medical patient can be described by several thousand characteristics, beginning with basic information such as age, weight, gender, but extending to lifestyle variables (diet, smoking, exercise) to an extensive array of variables describing someone's medical history.

Each of these problems has a multi-dimensional state vector, and in all but the last example the dimensions are continuous. In the car-rental example, the inventories will be discrete, but potentially fairly large (a major rental lot may have dozens of each type of car).

If we have 10 dimensions, and discretize each dimension into 100 elements, our input vector x (which might be a state) is $100^{10} = 10^{20}$ which is clearly a very large number. A

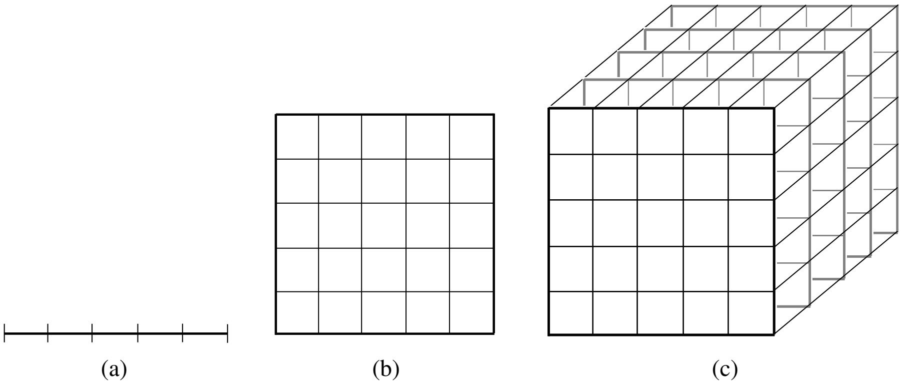


Figure 3.11 Illustration of the effect of higher dimensions on the number of grids in an aggregated state space.

reasonable strategy might be to aggregate. Instead of discretizing each dimension into 100 elements, what if we discretize into 5 elements? Now our state space is $5^{10} = 9.76 \times 10^6$, or almost 10 million states. Much smaller, but still quite large. Figure 3.11 illustrates the growth in the state space with the number of dimensions.

Each of our examples explode with the number of dimensions because we are using a lookup table representation for our function. It is important to realize that the curse of dimensionality is tied to the use of lookup tables. The other approximation architectures avoid the curse, but they do so by assuming structure such as a parametric form (linear or nonlinear).

Approximating high-dimensional functions is fundamentally intractable without exploiting structure. Beware of anyone claiming to "solve the curse of dimensionality." Pure lookup tables (which make no structural assumptions) are typically limited to five or six dimensions (depending on the number of values each dimension can take). However, we can handle thousands, even millions, of dimensions if we are willing to live with a linear model with separable, additive basis functions.

We can improve the accuracy of a linear model by adding features (basis functions) to our model. For example, if we use a second order parametric representation, we might approximate a two-dimensional function using

$$F(x) \approx \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_{11} x_1^2 + \theta_{22} x_2^2 + \theta_{12} x_1 x_2.$$

If we have N dimensions, the approximation would look like

$$F(x) \approx \theta_0 + \sum_{i=1}^{N} \theta_i x_i + \sum_{i=1}^{N} \sum_{j=1}^{N} \theta_{ij} x_i x_j,$$

which means we have to estimate $1 + N + N^2$ parameters. As N grows, this grows very quickly, and this is only a second order approximation. If we allow N^{th} order interactions, the approximation would look like

$$F(x) \approx \theta_0 + \sum_{i=1}^{N} \theta_i x_i + \sum_{i_1=1}^{N} \sum_{i_2=1}^{N} \theta_{i_1 i_2} x_{i_1} x_{i_2} + \sum_{i_1=1}^{N} \sum_{i_2=1}^{N} \cdots \sum_{i_N=1}^{N} \theta_{i_1, i_2, \ldots, i_N} x_{i_1} x_{i_2} \cdots x_{i_N}.$$

The number of parameters we now have to estimate is given by $1+N+N^2+N^3+\ldots+N^N$. Not surprisingly, this becomes intractable even for relatively small values of N.

The problem follows us if we were to use kernel regression, where an estimate of a function at a point s can be estimated from a series of observations $(\hat{f}^i, x^i)_{i=1}^N$ using

$$F(x) \approx \frac{\sum_{i=1}^N \hat{f}^i k(x, x^i)}{\sum_{i=1}^N k(x, x^i)}$$

where $k(x, x^i)$ might be the Gaussian kernel

$$k(x, x^i) = e^{-\frac{\|x-x^i\|^2}{b}}$$

where b is a bandwidth. Kernel regression is effectively a soft form of the aggregation depicted in figure 3.11(c). The problem is that we would have to choose a bandwidth that covers most of the data to get a statistically reliable estimate of a single point.

To see this, imagine that our observations are uniformly distributed in an N-dimensional cube that measures 1.0 on each side, which means it has a volume of 1.0. If we carve out an N-dimensional cube that measures .5 on a side, then this would capture 12.5 percent of the observations in a 3-dimensional cube, and 0.1 percent of the observations in a 10-dimensional cube. If we would like to choose a cube that captures $\eta = .1$ of our cube, we would need a cube that measures $r = \eta^{1/N} = .1^{1/10} = .794$, which means that our cube is covering almost 80 percent of the range of each input dimension.

The problem is that we have a multidimensional function, and we are trying to capture the joint behavior of all N dimensions. If we are willing to live with separable approximations, then we can scale to very large number of dimensions. For example, the approximation

$$F(x) \approx \theta_0 + \sum_{i=1}^N \theta_{1i} x_i + \sum_{i=1}^N \theta_{2i} x_i^2,$$

captures quadratic behavior but without any cross terms. The number of parameters is $1+2N$, which means we may be able to handle very high-dimensional problems. However, we lose the ability to handle interactions between different dimensions.

Kernel regression, along with essentially all nonparametric methods, is basically a fancy form of lookup table. Since these methods do not assume any underlying structure, they depend on capturing the local behavior of a function. The concept of "local," however, breaks down in high dimensions, where by "high" we typically mean three or more.

3.12.2 Designing approximation architectures in adaptive learning

Most solution methods in stochastic optimization are adaptive, which means that the data is arriving over time as a sequence of inputs x^n and observations $\hat{f}^{n+1}$. With each observation, we have to update our estimate of whatever function we are approximating, which might be the objective function $\mathbb{E}F(x, W)$, a value function $V(s)$, a policy $X^\pi(s)$, or a transition function $S^M(s, x, W)$. This entire chapter has focused on adaptive learning, but in the context where we used a fixed model and just adapt the parameters to produce the best fit.

Adaptive learning means that we have to start with small datasets (sometimes no data at all), and then adapt as new decisions and observations arrive. This raises a challenge we have not addressed above: we need to do more than just update a parameter vector θ^n with new data to produce θ^{n+1}. Instead, we need to update the architecture of the function we are trying to estimate. Said differently, the dimensionality of θ^n (or at least the set of nonzero elements of θ^n) will need to change as we acquire more data.

A key challenge with any statistical learning problem is designing a function that strikes the right tradeoff between the dimensionality of the function and the amount of data available for approximating the function. For a batch problem, we can use powerful tools such as regularization (see equation (3.54)) for identifying models that have the right number of variables given the available data. But this only works for batch estimation, where the size of the dataset is fixed.

As of this writing, additional research is needed to create the tools that can help to identify not just the best parameter vector θ^n, but the structure of the function itself. One technique that does this is hierarchical aggregation which we presented in the context of lookup tables in section 3.6. This is a powerful methodology that adaptively adjusts from a low dimensional representation (that is, estimates of the function at a high level of aggregation) to higher dimensional representations, which is accomplished by putting higher weights on the more disaggregate estimates. However, lookup table belief models are limited to relatively low dimensional problems.

3.13 WHY DOES IT WORK?**

3.13.1 Derivation of the recursive estimation equations

Here we derive the recursive estimation equations given by equations (3.57)-(3.61). To begin, we note that the matrix $(X^n)^T X^n$ is an $I+1$ by $I+1$ matrix where the element for row i, column j is given by

$$[(X^n)^T X^n]_{i,j} = \sum_{m=1}^{n} x_i^m x_j^m.$$

This term can be computed recursively using

$$[(X^n)^T X^n]_{i,j} = \sum_{m=1}^{n-1} (x_i^m x_j^m) + x_i^n x_j^n.$$

In matrix form, this can be written

$$[(X^n)^T X^n] = [(X^{n-1})^T X^{n-1}] + x^n (x^n)^T.$$

Keeping in mind that x^n is a column vector, $x^n(x^n)^T$ is an $I+1$ by $I+1$ matrix formed by the cross products of the elements of x^n. We now use the Sherman-Morrison formula (see section 3.13.2 for a derivation) for updating the inverse of a matrix

$$[A + uu^T]^{-1} = A^{-1} - \frac{A^{-1}uu^TA^{-1}}{1 + u^TA^{-1}u},$$

where A is an invertible $n \times n$ matrix, and u is an n-dimensional column vector. Applying this formula to our problem, we obtain

$$\begin{aligned}
[(X^n)^T X^n]^{-1} &= [(X^{n-1})^T X^{n-1} + x^n (x^n)^T]^{-1} \\
&= [(X^{n-1})^T X^{n-1}]^{-1} \\
&\quad - \frac{[(X^{n-1})^T X^{n-1}]^{-1} x^n (x^n)^T [(X^{n-1})^T X^{n-1}]^{-1}}{1 + (x^n)^T [(X^{n-1})^T X^{n-1}]^{-1} x^n}. \qquad (3.78)
\end{aligned}$$

The term $(X^n)^T Y^n$ can also be updated recursively using

$$(X^n)^T Y^n = (X^{n-1})^T Y^{n-1} + x^n(y^n). \tag{3.79}$$

To simplify the notation, let

$$\begin{aligned} B^n &= [(X^n)^T X^n]^{-1}, \\ \gamma^n &= 1 + (x^n)^T [(X^{n-1})^T X^{n-1}]^{-1} x^n. \end{aligned}$$

This simplifies our inverse updating equation (3.78) to

$$B^n = B^{n-1} - \frac{1}{\gamma^n}(B^{n-1} x^n (x^n)^T B^{n-1}).$$

Recall that

$$\bar{\theta}^n = [(X^n)^T X^n]^{-1} (X^n)^T Y^n. \tag{3.80}$$

Combining (3.80) with (3.78) and (3.79) gives

$$\begin{aligned} \bar{\theta}^n &= [(X^n)^T X^n]^{-1} (X^n)^T Y^n \\ &= \left(B^{n-1} - \frac{1}{\gamma^n}(B^{n-1} x^n (x^n)^T B^{n-1}) \right) \left((X^{n-1})^T Y^{n-1} + x^n y^n \right), \\ &= B^{n-1} (X^{n-1})^T Y^{n-1} \\ &\quad - \frac{1}{\gamma^n} B^{n-1} x^n (x^n)^T B^{n-1} \left[(X^{n-1})^T Y^{n-1} + x^n y^n \right] + B^{n-1} x^n y^n. \end{aligned}$$

We can start to simplify by using $\bar{\theta}^{n-1} = B^{n-1}(X^{n-1})^T Y^{n-1}$. We are also going to bring the term $x^n B^{n-1}$ inside the square brackets. Finally, we are going to bring the last term $B^{n-1} x^n y^n$ inside the brackets by taking the coefficient $B^{n-1} x^n$ outside the brackets and multiplying the remaining y^n by the scalar $\gamma^n = 1 + (x^n)^T B^{n-1} x^n$, giving us

$$\begin{aligned} \bar{\theta}^n &= \bar{\theta}^{n-1} - \frac{1}{\gamma^n} B^{n-1} x^n \left[(x^n)^T (B^{n-1} (X^{n-1})^T Y^{n-1}) \right. \\ &\quad \left. + (x^n)^T B^{n-1} x^n y^n - (1 + (x^n)^T B^{n-1} x^n) y^n \right]. \end{aligned}$$

Again, we use $\bar{\theta}^{n-1} = B^{n-1}(X^{n-1})^T Y^{n-1}$ and observe that there are two terms $(x^n)^T B^{n-1} x^n y^n$ that cancel, leaving

$$\bar{\theta}^n = \bar{\theta}^{n-1} - \frac{1}{\gamma^n} B^{n-1} x^n \left((x^n)^T \bar{\theta}^{n-1} - y^n \right).$$

We note that $(\bar{\theta}^{n-1})^T x^n$ is our prediction of y^n using the parameter vector from iteration $n-1$ and the explanatory variables x^n. y^n is, of course, the actual observation, so our error is given by

$$\hat{\varepsilon}^n = y^n - (\bar{\theta}^{n-1})^T x^n.$$

Let

$$H^n = -\frac{1}{\gamma^n} B^{n-1}.$$

We can now write our updating equation using

$$\bar{\theta}^n = \bar{\theta}^{n-1} - H^n x^n \hat{\varepsilon}^n. \tag{3.81}$$

3.13.2 The Sherman-Morrison updating formula

The Sherman-Morrison matrix updating formula (also known as the Woodbury formula or the Sherman-Morrison-Woodbury formula) assumes that we have a matrix A and that we are going to update it with the outer product of the column vector u to produce the matrix B, given by

$$B = A + uu^T. \tag{3.82}$$

Pre-multiply by B^{-1} and post-multiply by A^{-1}, giving

$$A^{-1} = B^{-1} + B^{-1}uu^T A^{-1}. \tag{3.83}$$

Post-multiply by u

$$\begin{aligned} A^{-1}u &= B^{-1}u + B^{-1}uu^T A^{-1}u \\ &= B^{-1}u\left(1 + u^T A^{-1}u\right). \end{aligned}$$

Note that $u^T A^{-1}u$ is a scalar. Divide through by $\left(1 + u^T A^{-1}u\right)$

$$\frac{A^{-1}u}{(1 + u^T A^{-1}u)} = B^{-1}u.$$

Now post-multiply by $u^T A^{-1}$

$$\frac{A^{-1}uu^T A^{-1}}{(1 + u^T A^{-1}u)} = B^{-1}uu^T A^{-1}. \tag{3.84}$$

Equation (3.83) gives us

$$B^{-1}uu^T A^{-1} = A^{-1} - B^{-1}. \tag{3.85}$$

Substituting (3.85) into (3.84) gives

$$\frac{A^{-1}uu^T A^{-1}}{(1 + u^T A^{-1}u)} = A^{-1} - B^{-1}. \tag{3.86}$$

Solving for B^{-1} gives us

$$\begin{aligned} B^{-1} &= [A + uu^T]^{-1} \\ &= A^{-1} - \frac{A^{-1}uu^T A^{-1}}{(1 + u^T A^{-1}u)}, \end{aligned}$$

which is the desired formula.

3.13.3 Correlations in hierarchical estimation

It is possible to derive the optimal weights for the case where the statistics $\bar{v}_s^{(g)}$ are not independent. In general, if we are using a hierarchical strategy and have $g' > g$ (which means that aggregation g' is more aggregate than g), then the statistic $\bar{v}_s^{(g',n)}$ is computed using observations $\hat{v}_s^n$ that are also used to compute $\bar{v}_s^{(g,n)}$.

We begin by defining

$$\begin{aligned}
\mathcal{N}_s^{(g,n)} &= \text{The set of iterations } n \text{ where } G^g(\hat{s}^n) = G^g(s) \text{ (that is, } \hat{s}^n \text{ aggregates to the same state as } s\text{).}\\
N_s^{(g,n)} &= |\mathcal{N}_s^{(g,n)}|\\
\bar{\varepsilon}_s^{(g,n)} &= \text{An estimate of the average error when observing state } s = G(\hat{s}^n).\\
&= \frac{1}{N_s^{(g,n)}} \sum_{n \in \mathcal{N}_s^{(g,n)}} \hat{\varepsilon}_s^{(g,n)}.
\end{aligned}$$

The average error $\bar{\varepsilon}_s^{(g,n)}$ can be written

$$\begin{aligned}
\bar{\varepsilon}_s^{(g,n)} &= \frac{1}{N_s^{(g,n)}} \left(\sum_{n \in \mathcal{N}_s^{(0,n)}} \varepsilon^n + \sum_{n \in \mathcal{N}_s^{(g,n)} \setminus \mathcal{N}_s^{(0,n)}} \varepsilon^n \right)\\
&= \frac{N_s^{(0,n)}}{N_s^{(g,n)}} \bar{\varepsilon}_s^{(0)} + \frac{1}{N_s^{(g,n)}} \sum_{n \in \mathcal{N}_s^{(g,n)} \setminus \mathcal{N}_s^{(0,n)}} \varepsilon^n. \qquad (3.87)
\end{aligned}$$

This relationship shows us that we can write the error term at the higher level of aggregation g' as a sum of a term involving the errors at the lower level of aggregation g (for the same state s) and a term involving errors from other states s'' where $G^{g'}(s'') = G^{g'}(s)$, given by

$$\begin{aligned}
\bar{\varepsilon}_s^{(g',n)} &= \frac{1}{N_s^{(g',n)}} \left(\sum_{n \in \mathcal{N}_s^{(g,n)}} \varepsilon^n + \sum_{n \in \mathcal{N}_s^{(g',n)} \setminus \mathcal{N}_s^{(g,n)}} \varepsilon^n \right)\\
&= \frac{1}{N_s^{(g',n)}} \left(N_s^{(g,n)} \frac{\sum_{n \in \mathcal{N}_s^{(g,n)}} \varepsilon^n}{N_s^{(g,n)}} + \sum_{n \in \mathcal{N}_s^{(g',n)} \setminus \mathcal{N}_s^{(g,n)}} \varepsilon^n \right)\\
&= \frac{N_s^{(g,n)}}{N_s^{(g',n)}} \bar{\varepsilon}_s^{(g,n)} + \frac{1}{N_s^{(g',n)}} \sum_{n \in \mathcal{N}_s^{(g',n)} \setminus \mathcal{N}_s^{(g,n)}} \varepsilon^n. \qquad (3.88)
\end{aligned}$$

We can overcome this problem by rederiving the expression for the optimal weights. For a given (disaggregate) state s, the problem of finding the optimal weights $(w_s^{(g,n)})_{g \in \mathcal{G}}$ is stated by

$$\min_{w_s^{(g,n)}, g \in \mathcal{G}} \mathbb{E} \left[\frac{1}{2} \left(\sum_{g \in \mathcal{G}} w_s^{(g,n)} \cdot \bar{v}_s^{(g,n)} - \nu_s^{(g,n)} \right)^2 \right] \qquad (3.89)$$

subject to

$$\begin{aligned}
\sum_{g \in \mathcal{G}} w_s^{(g,n)} &= 1 \qquad (3.90)\\
w_s^{(g,n)} &\geq 0, \quad g \in \mathcal{G}. \qquad (3.91)
\end{aligned}$$

Let

$$\begin{aligned}
\bar{\delta}_s^{(g,n)} &= \text{The error in the estimate } \bar{v}_s^{(g,n)} \text{ from the true value associated with} \\
&\quad\ \text{attribute vector } s. \\
&= \bar{v}_s^{(g,n)} - \nu_s.
\end{aligned}$$

The optimal weights are computed using the following theorem:

Theorem 1. *For a given attribute vector, s, the optimal weights, $w_s^{(g,n)}$, $g \in \mathcal{G}$, where the individual estimates are correlated by way of a tree structure, are given by solving the following system of linear equations in (w, λ):*

$$\sum_{g \in \mathcal{G}} w_s^{(g,n)} \mathbb{E}\left[\bar{\delta}_s^{(g,n)} \bar{\delta}_s^{(g',n)}\right] - \lambda = 0 \quad \forall\ g' \in \mathcal{G} \tag{3.92}$$

$$\sum_{g \in \mathcal{G}} w_s^{(g,n)} = 1 \tag{3.93}$$

$$w_s^{(g,n)} \geq 0 \quad \forall\ g \in \mathcal{G}. \tag{3.94}$$

Proof: The proof is not too difficult and it illustrates how we obtain the optimal weights. We start by formulating the Lagrangian for the problem formulated in (3.89)-(3.91), which gives us

$$\begin{aligned}
L(w,\lambda) &= \mathbb{E}\left[\frac{1}{2}\left(\sum_{g \in \mathcal{G}} w_s^{(g,n)} \cdot \bar{v}_s^{(g,n)} - \nu_s^{(g,n)}\right)^2\right] + \lambda\left(1 - \sum_{g \in \mathcal{G}} w_s^{(g,n)}\right) \\
&= \mathbb{E}\left[\frac{1}{2}\left(\sum_{g \in \mathcal{G}} w_s^{(g,n)}\left(\bar{v}_s^{(g,n)} - \nu_s^{(g,n)}\right)\right)^2\right] + \lambda\left(1 - \sum_{g \in \mathcal{G}} w_s^{(g,n)}\right).
\end{aligned}$$

The first order optimality conditions are

$$\mathbb{E}\left[\sum_{g \in \mathcal{G}} w_s^{(g,n)}\left(\bar{v}_s^{(g,n)} - \nu_s^{(g,n)}\right)\left(\bar{v}_s^{(g',n)} - \nu_s^{(g,n)}\right)\right] - \lambda = 0 \quad \forall\ g' \in \mathcal{G} \tag{3.95}$$

$$\sum_{g \in \mathcal{G}} w_s^{(g,n)} - 1 = 0. \tag{3.96}$$

To simplify equation (3.95), we note that,

$$\begin{aligned}
\mathbb{E}\left[\sum_{g \in \mathcal{G}} w_s^{(g,n)}\left(\bar{v}_s^{(g,n)} - \nu_s^{(g,n)}\right)\left(\bar{v}_s^{(g',n)} - \nu_s^{(g,n)}\right)\right] &= \mathbb{E}\left[\sum_{g \in \mathcal{G}} w_s^{(g,n)} \bar{\delta}_s^{(g,n)} \bar{\delta}_s^{(g',n)}\right] \\
&= \sum_{g \in \mathcal{G}} w_s^{(g,n)} \mathbb{E}\left[\bar{\delta}_s^{(g,n)} \bar{\delta}_s^{(g',n)}\right].
\end{aligned} \tag{3.97}$$

Combining equations (3.95) and (3.97) gives us equation (3.92) which completes the proof. □

Finding the optimal weights that handle the correlations between the statistics at different levels of aggregation requires finding $\mathbb{E}\left[\bar{\delta}_s^{(g,n)}\bar{\delta}_s^{(g',n)}\right]$. We are going to compute this expectation by conditioning on the set of attributes $\hat{s}^n$ that are sampled. This means that our expectation is defined over the outcome space Ω^ε. Let $N_s^{(g,n)}$ be the number of observations of state s at aggregation level g. The expectation is computed using:

Proposition 3.13.1. *The coefficients of the weights in equation (3.93) can be expressed as follows:*

$$\mathbb{E}\left[\bar{\delta}_s^{(g,n)}\bar{\delta}_s^{(g',n)}\right] = \mathbb{E}\left[\bar{\beta}_s^{(g,n)}\bar{\beta}_s^{(g',n)}\right] + \frac{N_s^{(g,n)}}{N_s^{(g',n)}}\mathbb{E}\left[\bar{\varepsilon}_s^{(g,n)^2}\right] \quad \forall g \leq g' \text{ and } g, g' \in \mathcal{G}. \tag{3.98}$$

The proof is given in section 3.13.4.

Now consider what happens when we make the assumption that the measurement error ε^n is independent of the attribute being sampled, $\hat{s}^n$. We do this by assuming that the variance of the measurement error is a constant given by ${\sigma_\varepsilon}^2$. This gives us the following result:

Corollary 3.13.1. *For the special case where the statistical noise in the measurement of the values is independent of the attribute vector sampled, equation (3.98) reduces to*

$$\mathbb{E}\left[\bar{\delta}_s^{(g,n)}\bar{\delta}_s^{(g',n)}\right] = \mathbb{E}\left[\bar{\beta}_s^{(g,n)}\bar{\beta}_s^{(g',n)}\right] + \frac{\sigma_\varepsilon^2}{N_s^{(g',n)}}. \tag{3.99}$$

For the case where $g = 0$ (the most disaggregate level), we assume that $\beta_s^{(0)} = 0$ which gives us

$$\mathbb{E}\left[\bar{\beta}_s^{(0,n)}\bar{\beta}_s^{(g',n)}\right] = 0.$$

This allows us to further simplify (3.99) to obtain

$$\mathbb{E}\left[\bar{\delta}_s^{(0,n)}\bar{\delta}_s^{(g',n)}\right] = \frac{\sigma_\varepsilon^2}{N_s^{(g',n)}}. \tag{3.100}$$

3.13.4 Proof of Proposition 3.13.1

We start by defining

$$\bar{\delta}_s^{(g,n)} = \bar{\beta}_s^{(g,n)} + \bar{\varepsilon}_s^{(g,n)}. \tag{3.101}$$

Equation (3.101) gives us

$$\begin{aligned}
\mathbb{E}\left[\bar{\delta}_s^{(g,n)}\bar{\delta}_s^{(g',n)}\right] &= \mathbb{E}\left[(\bar{\beta}_s^{(g,n)} + \bar{\varepsilon}_s^{(g,n)})(\bar{\beta}_s^{(g',n)} + \bar{\varepsilon}_s^{(g',n)})\right] \\
&= \mathbb{E}\left[\bar{\beta}_s^{(g,n)}\bar{\beta}_s^{(g',n)} + \bar{\beta}_s^{(g',n)}\bar{\varepsilon}_s^{(g,n)} + \bar{\beta}_s^{(g,n)}\bar{\varepsilon}_s^{(g',n)} + \bar{\varepsilon}_s^{(g,n)}\bar{\varepsilon}_s^{(g',n)}\right] \\
&= \mathbb{E}\left[\bar{\beta}_s^{(g,n)}\bar{\beta}_s^{(g',n)}\right] + \mathbb{E}\left[\bar{\beta}_s^{(g',n)}\bar{\varepsilon}_s^{(g,n)}\right] + \mathbb{E}\left[\bar{\beta}_s^{(g,n)}\bar{\varepsilon}_s^{(g',n)}\right] \\
&\quad + \mathbb{E}\left[\bar{\varepsilon}_s^{(g,n)}\bar{\varepsilon}_s^{(g',n)}\right].
\end{aligned} \tag{3.102}$$

We note that

$$\mathbb{E}\left[\bar{\beta}_s^{(g',n)}\bar{\varepsilon}_s^{(g,n)}\right] = \bar{\beta}_s^{(g',n)}\mathbb{E}\left[\bar{\varepsilon}_s^{(g,n)}\right] = 0.$$

Similarly

$$\mathbb{E}\left[\bar{\beta}_s^{(g,n)}\bar{\varepsilon}_s^{(g',n)}\right] = 0.$$

This allows us to write equation (3.102) as

$$\mathbb{E}\left[\bar{\delta}_s^{(g,n)}\bar{\delta}_s^{(g',n)}\right] = \mathbb{E}\left[\bar{\beta}_s^{(g,n)}\bar{\beta}_s^{(g',n)}\right] + \mathbb{E}\left[\bar{\varepsilon}_s^{(g,n)}\bar{\varepsilon}_s^{(g',n)}\right]. \tag{3.103}$$

We start with the second term on the right-hand side of equation (3.103). This term can be written as

$$\begin{aligned}\mathbb{E}\left[\bar{\varepsilon}_s^{(g,n)}\bar{\varepsilon}_s^{(g',n)}\right] &= \mathbb{E}\left[\bar{\varepsilon}_s^{(g,n)} \cdot \frac{N_s^{(g,n)}}{N_s^{(g')}}\bar{\varepsilon}_s^{(g,n)}\right] + \mathbb{E}\left[\bar{\varepsilon}_s^{(g,n)} \cdot \frac{1}{N_s^{(g')}} \sum_{n\in\mathcal{N}_s^{(g',n)}\setminus\mathcal{N}_s^{(g,n)}} \varepsilon^n\right] \\ &= \frac{N_s^{(g,n)}}{N_s^{(g')}}\mathbb{E}\left[\bar{\varepsilon}_s^{(g,n)}\bar{\varepsilon}_s^{(g,n)}\right] + \frac{1}{N_s^{(g')}}\underbrace{\mathbb{E}\left[\bar{\varepsilon}_s^{(g,n)} \cdot \sum_{n\in\mathcal{N}_s^{(g',n)}\setminus\mathcal{N}_s^{(g,n)}} \varepsilon^n\right]}_{I}..\end{aligned}$$

The term I can be rewritten using

$$\begin{aligned}\mathbb{E}\left[\bar{\varepsilon}_s^{(g,n)} \cdot \sum_{n\in\mathcal{N}_s^{(g',n)}\setminus\mathcal{N}_s^{(g,n)}} \varepsilon^n\right] &= \mathbb{E}\left[\bar{\varepsilon}_s^{(g,n)}\right]\mathbb{E}\left[\sum_{n\in\mathcal{N}_s^{(g',n)}\setminus\mathcal{N}_s^{(g,n)}} \varepsilon^n\right] \\ &= 0\end{aligned}$$

which means

$$\mathbb{E}\left[\bar{\varepsilon}_s^{(g,n)}\bar{\varepsilon}_s^{(g',n)}\right] = \frac{N_s^{(g,n)}}{N_s^{(g')}}\mathbb{E}\left[\bar{\varepsilon}_s^{(g)^2}\right]. \tag{3.104}$$

Combining (3.103) and (3.104) proves the proposition. □

The second term on the right-hand side of equation (3.104) can be further simplified using,

$$\begin{aligned}\mathbb{E}\left[\bar{\varepsilon}_s^{(g)^2}\right] &= \mathbb{E}\left[\left(\frac{1}{N_s^{(g,n)}}\sum_{n\in\mathcal{N}_s^{(g,n)}}\varepsilon^n\right)^2\right], \quad \forall\ g'\in\mathcal{G} \\ &= \frac{1}{\left(N_s^{(g,n)}\right)^2}\sum_{m\in\mathcal{N}_s^{(g,n)}}\sum_{n\in\mathcal{N}_s^{(g,n)}}\mathbb{E}\left[\varepsilon^m\varepsilon^n\right] \\ &= \frac{1}{\left(N_s^{(g,n)}\right)^2}\sum_{n\in\mathcal{N}_s^{(g,n)}}\mathbb{E}\left[(\varepsilon^n)^2\right] \\ &= \frac{1}{\left(N_s^{(g,n)}\right)^2}N_s^{(g,n)}{\sigma_\varepsilon}^2 \\ &= \frac{\sigma_\varepsilon^2}{N_s^{(g,n)}} \qquad (3.105)\end{aligned}$$

Combining equations (3.98), (3.104) and (3.105) gives us the result in equation (3.99). □

3.14 BIBLIOGRAPHIC NOTES

This chapter is primarily a brief tutorial into statistical learning. Readers interested in pursuing approximate dynamic programming should obtain a good statistical reference such as Bishop (2006) and Hastie et al. (2009). The second reference can be downloaded from

`http://www-stat.stanford.edu/~tibs/ElemStatLearn/`.

Sections 3.6 - Aggregation has been a widely used technique in dynamic programming as a method to overcome the curse of dimensionality. Early work focused on picking a fixed level of aggregation (Whitt (1978), Bean et al. (1987)), or using adaptive techniques that change the level of aggregation as the sampling process progresses (Bertsekas & Castanon (1989), Mendelssohn (1982), Bertsekas & Tsitsiklis (1996)), but which still use a fixed level of aggregation at any given time. Much of the literature on aggregation has focused on deriving error bounds (**?**, **?**). A recent discussion of aggregation in dynamic programming can be found in Lambert et al. (2002). For a good discussion of aggregation as a general technique in modeling, see Rogers et al. (1991). The material in section 3.6.3 is based on George et al. (2008) and Powell & George (2006). LeBlanc & Tibshirani (1996) and Yang (2001) provide excellent discussions of mixing estimates from different sources. For a discussion of soft state aggregation, see Singh et al. (1995). Section 3.5 on bias and variance is based on Powell & George (2006).

Section 3.7 - Basis functions have their roots in the modeling of physical processes. A good introduction to the field from this setting is Heuberger et al. (2005). Schweitzer & Seidmann (1985) describes generalized polynomial approximations for Markov decision processes for use in value iteration, policy iteration and the linear programming method. Menache et al. (2005) discusses basis function adaptations in the context of reinforcement learning. For a very nice discussion of the use of basis functions in approximate dynamic programming, see Tsitsiklis & Roy (1996) and Van Roy (2001). Tsitsiklis & Van Roy (1997) proves convergence of iterative stochastic algorithms for fitting the parameters of a regression model when the policy is held fixed. For section 18.4.1, the first use of approximate dynamic programming for evaluating an American call option is given in Longstaff & Schwartz (2001), but the topic has been studied for decades (see Taylor (1967)). Tsitsiklis & Van Roy (2001) also provide an alternative ADP algorithm for American call options. Clément et al. (2002) provides formal convergence results for regression models used to price American options. This presentation on the geometric view of basis functions is based on Tsitsiklis & Van Roy (1997).

Section **??** - There is, of course, an extensive literature on different statistical methods. This section provides only a sampling. For a much more thorough treatment, see Bishop (2006) and Hastie et al. (2009).

Section 3.10 - An excellent introduction to continuous approximation techniques is given in Judd (1998) in the context of economic systems and computational dynamic programming. Ormoneit & Sen (2002) and Ormoneit & Glynn (2002) discuss the use of kernel-based regression methods in an approximate dynamic programming setting, providing convergence proofs for specific algorithmic strategies. For a thorough introduction to locally polynomial regression methods, see **?**. An excellent discussion of a broad range of statistical learning methods can be found in Hastie et al. (2009). Bertsekas & Tsitsiklis (1996) provides an excellent discussion of neural networks in the context of approximate dynamic programming. Haykin (1999) presents a much more in-depth presentation of neural networks, including a chapter on approximate dynamic programming using neural networks. A very rich field of study has evolved around support vector machines and support vector regression. For a thorough tutorial, see Smola & Schölkopf (2004). A shorter and more readable introduction is contained in chapter 12 of Hastie et al. (2009). Note that SVR does not lend itself readily to recursive updating, which we suspect will limit its usefulness in approximate dynamic programming.

Section 3.12.1 - See Hastie et al. (2009), section 2.5, for a very nice discussion of the challenges of approximating high-dimensional functions.

PROBLEMS

3.1 This chapter is organized around three major classes of approximation architectures: lookup table, parametric and nonparametric, but some have argued that there should only be two classes: parametric and nonparametric. Justify your answer by presenting an argument why a lookup table can be properly modeled as a parametric model, and then a counter argument why a lookup table is more similar to a nonparametric model. [Hint: What is the defining characteristic of a nonparametric model? - see section 3.10.]

3.2 Use equations (3.30) and (3.31) to update the mean vector with prior

$$\bar{\mu}^0 = \begin{bmatrix} 10 \\ 18 \\ 12 \end{bmatrix}.$$

Assume that we test alternative 3 and observe $W = 19$ and that our prior covariance matrix Σ^0 is given by

$$\Sigma^0 = \begin{bmatrix} 12 & 4 & 2 \\ 4 & 8 & 3 \\ 2 & 3 & 10 \end{bmatrix}.$$

Assume that $\lambda^W = 4$. Give $\bar{\mu}^1$ and Σ^1.

3.3 In a spreadsheet, create a 4×4 grid where the cells are numbered 1, 2, ..., 16 starting with the upper left-hand corner and moving left to right, as shown below. We are going to treat each number in the cell as the mean of the observations drawn from that cell. Now assume that if we observe a cell, we observe the mean plus a random variable that is uniformly distributed between -1 and $+1$. Next define a series of aggregations where aggregation 0 is the disaggregate level, aggregation 1 divides the grid into four 2×2 cells, and aggregation 2 aggregates everything into a single cell. After n iterations, let $\bar{f}_s^{(g,n)}$ be the estimate of cell "s" at the n^{th} level of aggregation, and let

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

$$\bar{f}_s^n = \sum_{g \in \mathcal{G}} w_s^{(g)} \bar{f}_s^{(g,n)}$$

be your best estimate of cell s using a weighted aggregation scheme. Compute an overall error measure using

$$(\bar{\sigma}^2)^n = \sum_{s \in \mathcal{S}} (\bar{f}_s^n - \nu_s)^2,$$

where ν_s is the true value (taken from your grid) of being in cell s. Also let $w^{(g,n)}$ be the average weight after n iterations given to the aggregation level g when averaged over all cells at that level of aggregation (for example, there is only one cell for $w^{(2,n)}$). Perform 1000 iterations where at each iteration you randomly sample a cell and measure it with noise. Update your estimates at each level of aggregation, and compute the variance of your estimate with and without the bias correction.

(a) Plot $w^{(g,n)}$ for each of the three levels of aggregation at each iteration. Do the weights behave as you would expect? Explain.

(b) For each level of aggregation, set the weight given to that level equal to one (in other words, we are using a single level of aggregation) and plot the overall error as a function of the number of iterations.

(c) Add to your plot the average error when you use a weighted average, where the weights are determined by equation (3.46) without the bias correction.

(d) Finally add to your plot the average error when you used a weighted average, but now determine the weights by equation (3.47), which uses the bias correction.

(e) Repeat the above assuming that the noise is uniformly distributed between -5 and $+5$.

3.4 Show that

$$\sigma_s^2 = (\sigma_s^2)^{(g)} + (\beta_s^{(g)})^2 \tag{3.106}$$

which breaks down the total variation in an estimate at a level of aggregation is the sum of the variation of the observation error plus the bias squared.

3.5 In this exercise you will use the equations in section 3.8.1 to update a linear model. Assume you have an estimate of a linear model given by

$$\begin{aligned} \overline{F}(x|\theta^0) &= \theta_0 + \theta_1\phi_1(x) + \theta_2\phi_2(x) \\ &= -12 + 5.2\phi_1 + 2.8\phi_2. \end{aligned}$$

Assume that the matrix B^0 is a 3×3 identity matrix. Assume the vector $\phi = (\phi_0 \;\; \phi_1 \;\; \phi_2) = (5 \;\; 15 \;\; 22)$ and that you observe $\hat{f}^1 = 90$. Give the updated regression vector θ^1.

3.6 Show that $\mathbb{E}\left[\left(\bar{\mu}^{n-1} - \mu(n)\right)^2\right] = \lambda^{n-1}\sigma^2 + (\beta^n)^2$ (equation (3.38)). [Hint: Add and subtract $\mathbb{E}\bar{\mu}^{n-1}$ inside the expectation and expand.]

3.7 Show that $\mathbb{E}\left[\left(\bar{\theta}^{n-1} - \hat{\theta}^n\right)^2\right] = (1 + \lambda^{n-1})\sigma^2 + (\beta^n)^2$ (which proves equation 3.39). [Hint: See previous exercise.]

3.8 Derive the small sample form of the recursive equation for the variance given in (3.40). Recall that if

$$\bar{\mu}^n = \frac{1}{n}\sum_{m=1}^{n} \hat{\mu}^m$$

then an estimate of the variance of $\hat{\theta}$ is

$$Var[\hat{\mu}] = \frac{1}{n-1}\sum_{m=1}^{n} (\hat{\mu}^m - \bar{\mu}^n)^2.$$

PART II - LEARNING PROBLEMS

Learning problems represent a broad class of problems that are typically grouped under names such as stochastic search, ranking and selection, simulation optimization, and multiarmed bandit problems. We include in this part problems that are often solved using iterative algorithms, where the only information carried from one iteration to the next is what we have learned about the function. This is the defining characteristic of a learning problem.

We begin in chapter 4 by providing an overview of learning problems that arise in stochastic optimization. This chapter reminds us that there are special cases of problems that can be solved exactly, possibly by replacing the original expectation with a sampled approximation. The chapter closes by setting up some basic concepts for learning problems, including making the important distinction between online and offline problems, and by identifying different strategies for designing policies for adaptive learning.

Chapter 5 begins with derivative-based algorithms, where we describe the difference between asymptotic and finite-time analysis. This chapter identifies the importance of stepsizes, which are actually "decisions" in derivative-based methods. Chapter 6 provides an in-depth discussion of stepsize policies.

We then transition to derivative-free problems in chapter 7, where there is a much richer tradition of designing policies (compared to derivative-based methods).

By the end of Part II, we will have laid the foundation for the much richer class of sequential decision problems that involve controllable physical states that link decisions and dynamics from one time period to the next.

CHAPTER 4

INTRODUCTION TO STOCHASTIC OPTIMIZATION

Our most basic optimization problem has the core structure: make decision, learn information, stop. This can be written

$$\max_{x\in\mathcal{X}} \mathbb{E}F(x,W), \tag{4.1}$$

where W is any form of random variable. This is sometimes called the *static* stochastic optimization problem, because it consists of making a single decision x, then observing an outcome W allowing us to assess the performance $F(x,W)$, at which point we stop. This contrasts with fully sequential problems where, for example, we have to manage inventories over time, consisting of a series of decisions (orders), followed by information (demands), after which we place a new order, and so on.

While equation (4.1) is the most standard way of writing this problem, we are going to use as our default statement

$$\max_{x\in\mathcal{X}} \mathbb{E}\{F(x,W)|S_0\}, \tag{4.2}$$

which allows us to express the expectation on information in an initial state S_0, which can include deterministic parameters as well as probabilistic information (which we need when we use Bayesian belief models). For example, our problem may depend on an unknown physical parameter $\boldsymbol{\theta}$ which we believe may be one of a set $\theta_1,\ldots,\theta_K$ with probability $p_k^0 = \mathbb{P}[\boldsymbol{\theta} = \theta_k]$.

There are three core strategies for solving our basic stochastic optimization problem: (4.2):

Deterministic methods - There are some problems with sufficient structure that allows us to solve the problem deterministically. In some cases problems can be solved analytically, while others require the use of deterministic optimization algorithms.

Sampled approximations - This is a powerful and widely used approach for turning computationally intractable expectations into tractable ones. We note that sampled problems, while solvable, may not be easily solvable, and as a result have attracted considerable research interest (especially for problems where x is high-dimensional, and especially when it is integer).

Adaptive learning methods - These approaches, which will attract most of our attention in this volume, use iterative methods to solve sequences of relatively easy problems, with the hope that the algorithms will converge to the solution of the original problem.

We begin our presentation by discussing different perspectives of our basic stochastic optimization problem, which encompasses fully sequential problems when we interpret "x" as a policy π. We then observe that there are examples of stochastic optimization problems that can be solved using standard deterministic methods, either by directly exploiting the structure of the uncertainty (which allows us to compute the expectation directly), or by using the powerful idea of sampled models.

We then close by setting up some preliminary discussions about adaptive learning methods, which are then discussed in more detail in chapters 5 and 7. As we point out below, adaptive learning methods represent a form of sequential decision problem where the state variable S^n captures only what we know. There is no other physical process (such as inventory) or informational process (such as a time series) which links decisions over time. We defer until Part III of the book the handling of these more complex problems.

4.1 ILLUSTRATIONS OF THE BASIC STOCHASTIC OPTIMIZATION PROBLEM

There is no shortage of applications of our basic stochastic optimization problem. Some examples that illustrate applications in different settings include:

■ **EXAMPLE 4.1**

Engineering design - Here x is the design of an airplane wing where we have to create a design that minimizes costs over a range of different conditions. We can learn from numerical simulations, laboratory strength tests, and examining actual aircraft for stress fractures.

■ **EXAMPLE 4.2**

Let $(y^n, x^n)_{n=1}^N$ be a set of explanatory variables x^n and response variables y^n. We would like to fit a statistical model (this might be a linear parametric model, or a neural network) where θ is the parameters (or weights) that characterize the model. We want to find θ that solves

$$\min_\theta \frac{1}{N}\sum_{n=1}^N (y^n - f(x^n|\theta))^2.$$

This problem, which is very familiar in statistics, is a sampled approximation of

$$\min_\theta \mathbb{E}(Y - f(X|\theta))^2,$$

where X is a random input and Y is the associated random response.

■ **EXAMPLE 4.3**

We would like to design an energy system where R is a vector of energy investments (in wind farms, solar fields, battery storage, gas turbines), which we have to solve subject to random realizations of energy from wind and solar (which we represent using the vector W) defined over a year. Let $C^{cap}(R)$ be the capital cost of these investments, and let $C^{op}(R, W)$ be the net operating revenue given W (computed from a numerical simulator). Now we want to solve

$$\max_R \mathbb{E}(-C^{cap}(R) + C^{op}(R, W)).$$

■ **EXAMPLE 4.4**

A bank uses a policy $X^\pi(S|\theta)$ that covers how much to move into or out of cash given the state S which describes how much cash is on hand, the forward price/earnings ratio of the S&P 500 (an important index of the stock market), and current 10-year bond rates. The vector θ captures upper and lower limits on each variable that triggers decisions to move money into or out of cash. If $C(X^\pi(S_t|\theta), W_{t+1})$ is the cash flow given the current state S_t and the next-period returns W_{t+1}, then we want to find the policy control parameters θ that solves

$$\max_\theta \mathbb{E} \sum_{t=0}^{T} e^{-rt} C(X^\pi(S_t|\theta), W_{t+1}).$$

Each of these examples involve making some decision (the design of the airplane wing, the model parameter θ, the energy investment R, or the parameters θ of a cash transfer policy). In each case, we have to choose a design either to optimize a deterministic function, a sampled approximation of a stochastic problem, or by adaptive learning (either from a simulator, laboratory experiments or field observations).

While there are some settings where we can solve (4.2) (possibly with an approximation of the expectation) directly, most of the time we are going to turn to iterative learning algorithms. We will start with a state S^n that captures our belief state about the function $F(x) = \mathbb{E}\{F(x, W)|S_0\}$ after n repetitions. We then use this knowledge to make a decision x^n after which we observe W^{n+1} which leads us to a new belief state S^{n+1}. The problem is designing a good rule (or policy) that we call $X^\pi(S^n)$ that determines x^n. For example, we might want to find the best answer that we can with a budget of N iterations. We pose this as one of finding the best policy to determine a solution $x^{\pi,N}$, which would be formulated as

$$\max_\pi \mathbb{E}\{F(x^{\pi,N}, W)|S_0\}. \tag{4.3}$$

The formulations in (4.1), (4.2) and (4.3) all focus on finding the best decision (or design) to maximize some function. We refer to these as *terminal reward* formulations. This distinction is important when we use adaptive learning policies $X^\pi(S)$, since this involves optimizing using intelligent trial and error.

When we use adaptive learning (which is a widely used strategy), then we have to think about our attitude toward the intermediate decisions x^n for $n < N$. If we have to "count" the results of these intermediate experiments, then we would write our objective as

$$\max_\pi \mathbb{E} \sum_{n=0}^{N-1} \{F(X^\pi(S^n), W^{n+1}) | S_0\}. \tag{4.4}$$

When we are using an adaptive learning strategy, we are going to refer to (4.3) as the *terminal reward* formulation, while the objective function in (4.4) is the *cumulative reward* formulation.

On first glance it appears that our adaptive optimization problem with cumulative rewards in (4.4) is quite different from our original basic stochastic optimization problem in (4.2), but take another look. If we assume we can represent our policy π as x (and we will), and then simply let $F(x, W)$ be the summation in (4.4), then we come full circle.

The number of applications that fit the basic model given in equation (4.2) is limitless. For discussion purposes, it is helpful to recognize some of the major problem classes that arise in this setting:

- Discrete problems, where $\mathcal{X} = \{x_1, \ldots, x_M\}$. Examples might be where x_m is a set of features for a product, catalysts for a type of material, drug cocktails, or even paths over a network.
- Concave problems, where $F(x, W)$ is concave in x.
- Linear programs, where $F(x, W)$ is a linear cost function and $\mathcal{X}$ is a set of linear constraints.
- Continuous, nonconcave problems, where x is continuous.
- Expensive functions - There are many settings where computing $F(x, W)$ involves running time consuming computer simulations or laboratory experience that may take hours to days to weeks, or field experiments that may take weeks or months).

For these problems, the decision x may be finite, continuous scalar, or a vector (that may be continuous or integer).

As we progress, we are going to see many instances of (4.1) (or (4.2)) where we sequentially guess at a decision x^n, then observe W^{n+1}, and use this information to make a better guess x^{n+1}, with the goal of solving (4.1). In fact, before we are done, we are going to show that we can reduce our formulations of fully sequential problems such as our inventory problem to the same form as in (4.1). For this reason, we have come to refer to (4.2) as the basic stochastic optimization model.

4.2 DETERMINISTIC METHODS

There are a handful of stochastic optimization problems that can be solved to optimality using purely deterministic methods. We are going to provide a brief illustration of some examples as an illustration, but in practice, exact solutions of stochastic problems will be quite rare.

4.2.1 A "stochastic" shortest path problem

In section 8.1.1, we introduced a stochastic shortest path problem where a traveler arriving to node i would see the sample realizations of the random costs C_{ij} to each node j that can be reached from i. Assume that on the nth day we arrive to node i and observe the sample realization $\hat{c}_{ij}^n$ of the random variable C_{ij}. We would then get a sampled observation of the value of being at node i from

$$\hat{v}_i^n = \min_{j \in \mathcal{I}_i^+} \big(\hat{c}_{ij}^n + \overline{V}_j^{n-1}\big),$$

where $\mathcal{I}_i^+$ is the set of all nodes that we can reach from node i. Now assume that we do not see the sample realization of the random variable C_{ij} before we make our decision. Assume we have to make the decision before we see the realization. In this case, we have to use the expected value $\bar{c}_{ij} = \mathbb{E}C_{ij}$, which means we are solving

$$\begin{aligned} \hat{v}_i^n &= \min_{j \in \mathcal{I}_i^+} \mathbb{E}\big(C_{ij} + \overline{V}_j^{n-1}\big), \\ &= \min_{j \in \mathcal{I}_i^+} \big(\bar{c}_{ij} + \overline{V}_j^{n-1}\big), \end{aligned}$$

which is just what we would solve if we had a deterministic shortest path problem. In other words, when we have a linear objective, if we have to make decisions before we see information, then resulting problem reduces to a deterministic optimization problem which can (generally) be solved exactly.

The key difference between this "stochastic" shortest path problem and the one in section 8.1.1 is how information is revealed. The problem in section 8.1.1 is harder (and more interesting) because information is revealed just before we make the decision of the next link to traverse. Here, information is revealed after we make a decision, which means decisions have to be made using distributional information. Since the problem is linear in the costs, then all we need are the means, turning our stochastic problem into a deterministic problem.

4.2.2 A newsvendor problem with known distribution

We next consider one of the oldest stochastic optimization problems, known as the newsvendor problem, which is given by

$$\max_x EF(x, W) = \mathbb{E}\big(p \min\{x, W\} - cx\big). \tag{4.5}$$

Assume that we know the cumulative distribution $F^W(w) = \mathbb{P}[W \leq w]$ of the demand W. We begin by computing the stochastic gradient, given by

$$\nabla_x F(x, W) = \begin{cases} p - c & \text{If } x \leq W \\ -c & \text{If } x > W. \end{cases} \tag{4.6}$$

We next observe that if $x = x^*$, the optimal solution, then the expectation of the gradient should be zero. This means

$$\begin{aligned} \mathbb{E}\nabla_x F(x, W) &= (p - c)\mathbb{P}[x^* \leq W] - c\mathbb{P}[x^* > W], \\ &= (p - c)\mathbb{P}[x^* \leq W] - c(1 - \mathbb{P}[x^* \leq W]), \\ &= 0. \end{aligned}$$

Solving for $\mathbb{P}[x^* \leq W]$ gives

$$\mathbb{P}[x^* \leq W] = \frac{c}{p}. \tag{4.7}$$

Under the (reasonable) assumption that the unit purchase cost c is less than the sales price p, we see that the optimal solution x^* corresponds to the point where the probability that x^* is less than the demand W is the ratio of the cost over the price. Thus if the cost is low, the probability that the demand is greater than the supply (which means we lose sales) should be low.

Equation (4.7) gives the optimal solution of the newsvendor problem. It requires that we know the distribution of demand, and also requires that we be able to take the expectation of the gradient and solve for the optimal probability analytically. Not surprisingly, these conditions are rarely met in practice.

4.2.3 Chance constrained optimization

There are some problems where we can compute the expectation exactly, but the result is (typically) a nonlinear problem that can only be solved numerically. A good illustration of this is a method known as chance constrained programming, which is itself a rich area of study. A classical formulation (which we first saw in section 2.1.7) poses the problem

$$\min_x f(x), \tag{4.8}$$

subject to the constraint

$$p(x) \quad \leq \quad \alpha, \tag{4.9}$$

where

$$p(x) \quad = \quad \mathbb{P}[C(x,W) \geq 0] \tag{4.10}$$

is the probability that a constraint violation, captured by $C(x, W)$, is violated. Thus, $C(x, W)$ might be the uncovered demand for energy, or the degree to which two driverless cars get closer than an allowed tolerance. If we can compute $p(x)$ (analytically or numerically), we can draw on powerful nonlinear programming algorithms to solve (4.8) directly.

4.2.4 Optimal control

In section 2.1.11, we formulated an optimal control problem of the form

$$\min_{u_0,\ldots,u_T} \sum_{t=0}^{T} L_t(x_t, u_t).$$

where states evolve according to $x_{t+1} = f(x_t, u_t)$. We may introduce a stochastic noise term giving us the state transition equation

$$x_{t+1} = f(x_t, u_t) + w_t,$$

where (following the standard convention of the controls community) w_t is random at time t. The historical basis for this notational convention is the roots of optimal control in continuous time, where w_t would represent the noise between t and $t + dt$. In the presence of noise, we need to introduce a policy $U^\pi(x_t)$. We would now write our objective function as

$$\min_\pi \mathbb{E} \sum_{t=0}^{T} L_t(x_t, U_t^\pi(x_t)). \tag{4.11}$$

Now assume that the loss function has the quadratic form

$$L_t(x_t, u_t) = (x_t)^T Q_t x_t + (u_t)^T R_t u_t.$$

After quite a bit of algebra, it is possible to show that the optimal policy has the form

$$U_t^\pi(x_t) = K_t x_t, \tag{4.12}$$

where K_t is a complex matrix that depends on the matrices Q_t and R_t.

This solution depends on three critical features of this problem:

- The objective function is quadratic in the state x_t and the control u_t.
- The control u_t is unconstrained.
- The noise term w_t is additive in the transition function.

Despite these limitations, this result has proved quite important for many problems in engineering.

4.2.5 Discrete Markov decision processes

As with the field of stochastic control, there is an incredibly rich body of literature that has grown up around the basic problem of discrete dynamic programs, a problem that we address in much more depth in chapter 14. Imagine that we have a contribution $C(s, a)$ when we are in state $s \in \mathcal{S}$ and take action $a \in \mathcal{A}$, and a one-step transition matrix $p(s'|s, a)$ which gives the probability that we evolve to state $S_{t+1} = s'$ given that we are in state $S_t = s$ and take action a. It is possible to show that the value of being in a state $S_t = s$ at time t is given by

$$V_t(S_t) = \max_{a \in \mathcal{A}} \left(C(S_t, a) + \sum_{s' \in \mathcal{S}} p(s'|S_t, a) V_{t+1}(s') \right). \tag{4.13}$$

We can compute (4.13) if we start at time T with some initial value, say $V_T(s) = 0$, and then step backward in time. This produces the optimal policy $A_t^*(S_t)$ given by

$$A_t^*(S_t) = \arg\max_{a \in \mathcal{A}} \left(C(S_t, a) + \sum_{s' \in \mathcal{S}} p(s'|S_t, a) V_{t+1}(s') \right). \tag{4.14}$$

Again, we have found our optimal policy purely using deterministic mathematics. The critical element of this formulation is the assumption that the one-step transition matrix $p(s'|s, a)$ is known (and computable). This requirement also requires that the state space $\mathcal{S}$ and action space $\mathcal{A}$ be discrete and not too large.

4.2.6 Remarks

These are a representative list of the very small handful of stochastic optimization problems that can be solved either analytically or numerically, using deterministic methods. While we have not covered every problem that can be solved this way, the list is not long. This is not to minimize the importance of these results, which sometimes serve as the foundation for algorithms for more general problems.

Often, the most difficult aspect of a stochastic optimization problem is the expectation (or other operators such as risk metrics to deal with uncertainty). It should not be surprising, then, that the techniques used to solve more general stochastic optimization problems tend to focus on simplifying or breaking down the representation of uncertainty. The next section introduces the concept of sampled models, a powerful strategy that is widely used in stochastic optimization. We then transition to a discussion of adaptive sampling-based methods that is the focus of most of the rest of this book.

4.3 SAMPLED MODELS

One of the most powerful and widely used methods in stochastic optimization is to replace the expectation in the original model in equation (4.1), which is typically computationally intractable, with a sampled model. The idea overcomes one of the problems of stochastic optimization, which is that expectations, which are so easy to write, are so often very difficult to compute.

Our newsvendor problem is a nice example of a stochastic optimization problem where the uncertain random variable is a scalar. This means that we can write the expectation as an integral (if the random variable is continuous) or perhaps a sum, if the random variable is discrete or if it can be discretized. However, neither approach works when the random variable is a vector, and there are problems where random variables may have dozens or hundreds of dimensions (such as the random prices of stocks in a portfolio) or even tens of thousands of dimensions (the random flows of customers or freight between pairs of locations around a city or the country). A few examples illustrate how large random variables can get to be:

■ **EXAMPLE 4.1**

A blood management problem requires managing eight blood types, which can be anywhere from 0 to 5 weeks old, and may or may not be frozen, creating $6\times 8\times 2 = 96$ blood types. Patients needing blood create demands for eight different types of blood. Each week there are random supplies (96 dimensions) and random demands (8 dimensions), creating an exogenous information variable W_t with 104 dimensions.

■ **EXAMPLE 4.2**

A freight company is moving parcels among 1,000 different terminals. Since each parcel has an origin and destination, the vector of new demands has 1,000,000 dimensions.

■ **EXAMPLE 4.3**

Patients arriving to a doctor's office may exhibit as many as 300 different characteristics. Since each patient may or may not have any of these characteristics, there are as many as $2^{300} \sim 2 \times 10^{90}$ different types of patients.

This section provides a brief introduction to what has evolved into an incredibly rich literature. We start by addressing the following questions:

- How do we formulate a sampled model?
- How good is the quality of the sampled solution (and how fast does it approach the optimal as N is increased)?
- For large problems (high dimensional x), what are strategies for solving (4.15)?
- Again for large problems, what are the best ways of creating the sample $W^1, \ldots, W^N$?

We are going to return to sampled models from time to time since they represent such a powerful strategy for handling expectations.

4.3.1 Formulating a sampled model

Assume that W is one of these multidimensional (and possibly very high dimensional) random variables. Further assume that we have some way of generating a set of samples $W^1, \ldots, W^N$. These may be generated from a known probability distribution, or perhaps from a historical sample. We can replace our original stochastic optimization problem (4.1) with

$$\max_x \frac{1}{N} \sum_{n=1}^{N} F(x, W^n). \tag{4.15}$$

Solving (4.15) as an approximation of the original problem in (4.1) is known as the *sample average approximation*. It is important to realize that both our original stochastic optimization problem (4.1) and the sampled problem (4.15) are deterministic optimization problems. The challenge is computation.

Below we illustrate several uses of sampled models.

4.3.1.1 A sampled stochastic linear program As with W, the decision variable x can be a scalar, or a very high-dimensional vector. For example, we might have a linear program where we are optimizing the flows of freight x_{ij} from location i to location j by solving

$$\min_x F(x, W) = \sum_{i,j \in \mathcal{I}} c_{ij} x_{ij},$$

subject to a set of linear constraints

$$\begin{aligned} Ax &= b, \\ x &\geq 0. \end{aligned}$$

Now assume that the random information is the cost vector c (which might reflect transportation costs which depend on traffic congestion) and the vector b, which reflects all the demands for travel. Thus, $W = (A, b, c)$. If we have one sample of W, then we have a straightforward linear program which may not be too hard to solve. But now imagine that we have $N = 100$ samples of the data, given by $(A^n, b^n, c^n)_{n=1}^N$. we could then solve

$$\min_x \frac{1}{N} \sum_{n=1}^{N} c_{ij}^n x_{ij},$$

subject to, for $n = 1, \ldots, 100$:

$$\begin{aligned} A^n x &= b^n, \\ x &\geq 0. \end{aligned}$$

If we choose a sample of $N = 100$ samples, then our sampled problem in (4.15) becomes a linear program that is 100 times larger. This may be computationally difficult (in fact, coming up with a single vector x that is feasible for all 100 samples of the data (A, b, c) may not even be possible). A common application of this strategy arises when making a decision to allocate a resource such as blood inventories from central blood inventories to hospitals, before knowing the results of weekly donations of blood, and the schedule of operations that need blood, at each hospital for the following week.

4.3.1.2 Sampled chance constrained models

We can use our idea of sampling to solve chance constrained programs. We begin by noting that a probability is like an expectation. Let $\mathbb{1}_{\{E\}} = 1$ if event E is true. Then we can write our probability as

$$\mathbb{P}[C(x, W) \leq 0] = \mathbb{E}\mathbb{1}_{\{C(x,W) \leq 0\}}.$$

We can replace the chance constraint in (4.10) with a sampled version, where we basically average the random indicator variable to obtain

$$\mathbb{P}[C(x, W) \leq 0] \approx \frac{1}{N} \sum_{n=1}^{N} \mathbb{1}_{\{C(x,W^n) \leq 0\}}.$$

If x is discrete, then each $\mathbb{1}_{\{C(x,W^n)\}}$ can be calculated in advance for each W^n. If x is continuous, then it is likely that these indicator functions can be written as linear constraints.

4.3.1.3 Sampled parametric models

Sampled models may take other forms. Imagine that we wish to model demand as a function of price using a logistic function

$$D(p|\theta) = D^0 \frac{e^{\theta_0 - \theta_1 p}}{1 + e^{\theta_0 - \theta_1 p}}.$$

We want to pick a price that maximizes revenue using

$$R(p|\theta) = pD(p|\theta).$$

Our problem is that we do not know θ. We might assume that our vector θ follows a multivariate normal distribution, in which case we would want to solve

$$\max_p \mathbb{E}_\theta pD(p|\theta), \tag{4.16}$$

but computing the expectation may be hard. However, perhaps we are willing to say that θ may take on one of a set of values $\theta_1, \ldots, \theta_K$, each with probability q_k. Now we can solve

$$\max_p \sum_{k=1}^{K} pD(p|\theta_k)q_k. \tag{4.17}$$

Whereas equation (4.16) may be intractable, (4.17) may be much easier.

Both (4.15) and (4.17) are examples of sampled models. However, the representation in (4.15) is used in settings where W^n is a sample drawn from a typically large (often infinite) set of potential outcomes, where each is drawn with equal likelihood. The model in (4.17) is used when we have an uncertain belief about parameters, and are using the set $\theta_1, \ldots, \theta_K$, with a probability vector q that may evolve over time. For now, we are going to assume that we have drawn a random sample $W^1, \ldots, W^N$ where each is sampled with equal likelihood.

4.3.2 Benders decomposition for a sampled convex problem

There are many problems where $F(x, W)$ is convex in x for a given value of W. An important problem class is a variant of our basic stochastic optimization problem known as the two-stage stochastic programming problem. In this problem, instead of making a single decision x and then observing W, we make an initial decision x_0, then observe W_1, and finally get to make one more decision x_1. Both x_0 and x_1 are vectors subject to linear constraints. For example, imagine that x_0 involves the decision on how much inventory Amazon should place in each of its fulfillment centers. Then, after observing the $W_1 = (D_1, c_1)$, where D_1 is the demand for various products and c_1 are the transportation costs, Amazon then has to decide from which fulfillment centers demands should be satisfied.

The problem to find x_0, known as the first stage problem, is given by

$$\max_{x_0} \big(c_0 x_0 + \mathbb{E} Q_1(x_0, W)\big). \tag{4.18}$$

This problem is solved subject to the constraints,

$$\begin{aligned} A_0 x_0 &= b, && (4.19)\\ x_0 &\geq 0, && (4.20) \end{aligned}$$

where (4.19) represents constraints on how much inventory can be placed in each fulfillment center (captured by b). We then solve the second stage problem to determine x_1, given the first stage decisions. Assume that we observe outcome ω for the random variable W. The resulting problem would be written

$$Q_1(x_0, \omega) = \max_{x_1(\omega)} c_1(\omega) x_1(\omega), \tag{4.21}$$

subject to, for all $\omega \in \Omega$,

$$\begin{aligned} A_1 x_1(\omega) &\leq B_1 x_0, && (4.22)\\ B_1 x_1(\omega) &\leq D_1(\omega), && (4.23)\\ x_1(\omega) &\geq 0. && (4.24) \end{aligned}$$

Equation (4.22) enforces the constraints that we cannot ship product from fulfillment centers that have not been built (and we have to stay within the capacity of centers that have been built). Equation (4.23) represents the demand constraints, where we assume our contribution vector c_1 is designed to give a high incentive to meet demand. Let $\beta(\omega)$ be the dual variable of the resource constraint (4.22) which reflects the effect of the first stage decision x_0 on the second stage. The function $Q_1(x_0, W)$ is known in the stochastic programming literature as the *recourse function* since it allows us to respond to different outcomes using the variables $x_1(\omega)$ which are chosen after choosing x_1 and observing $W(\omega)$. Thus, we might want to satisfy demand in Texas from a nearby fulfillment center in Houston, but if that center does not have sufficient inventory, our *recourse* is to satisfy demand from a more distant center in Chicago.

We face the challenge of approximating the function $Q_1(x_0) = \mathbb{E}Q_1(x_0, W)$ so that we can solve the initial problem for x_0 in equation (4.18). It would also be nice if we could do this in a way so that we can solve the first stage problem as a linear program, which makes it easy to handle the vector x_0. There are several strategies we can draw on, but here we are going to illustrate a powerful idea known as Benders decomposition. In a nutshell, our second stage function $Q_1(x_0, W)$ is a linear program, which means that it is concave in the right hand side constraint $B_1 x_0$ (because we are maximizing).

We illustrate Benders decomposition in the context of solving a sampled version of the problem. We do this by replacing our original full sample space Ω (over which the original expectation $\mathbb{E}$ is defined) with a sampled set of outcomes $\hat{\Omega} = (\omega_1, \ldots, \omega_K)$. For each solution, we would obtain the optimal value $\hat{Q}(x_0, \omega)$, and the corresponding dual variable $\beta(\omega)$. We then define the sampled Q-function as

$$\hat{Q}_1(x_0) = \frac{1}{K} \sum_{k=1}^{K} Q_1(x_0, \omega^k).$$

Benders decomposition iteratively builds up an approximation of $Q_1(x_0)$ by constructing a series of *supporting hyperplanes* (we could also call these linear approximations) derived by solving the second stage linear program for individual samples ω of the random vector W. We do this by solving equations (4.21)-(4.24) for $\omega = \omega^k$ for $k = 1, \ldots, K$, and obtain

$$\begin{aligned} \alpha^k &= Q(x_0, \omega^k), \\ \beta^k &= \beta(\omega^k). \end{aligned}$$

where $\beta(\omega^k)$ is the dual variable for constraint (4.22). We then solve

$$x_0^* = \arg\max_{x_0, z} \big(c_0 x_0 + z\big), \tag{4.25}$$

subject to (4.19) - (4.20) and

$$z \leq \alpha^k + \beta^k x_0, \;\; k = 1, \ldots, K. \tag{4.26}$$

Equation (4.26) creates a multidimensional envelop, depicted in figure 4.1, which depicts the sampled function $\hat{Q}_1(x_0)$ and the original true function $Q_1(x_0)$. Note that the hyperplanes touch the sampled function $\hat{Q}_1(x_0)$, but only approximate the true function $Q_1(x_0)$.

We close by noting that this is one way of solving convex problems, but it requires assuming that the sampled approximation will provide a good solution. This has opened a body of literature focusing on the design of good samples, which is challenging in the high dimensional settings of linear programs. Later, we are going to revisit this approach using an adaptive method which iteratively samples from the full space.

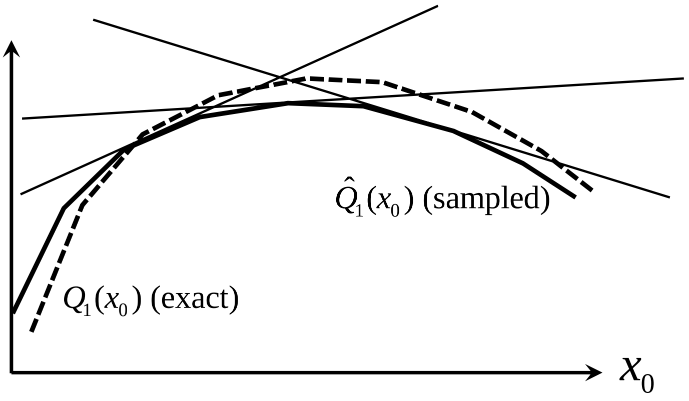


Figure 4.1 Illustration of Benders cuts shown next to exact ($Q_1(x_0)$) and sampled ($\hat{Q}_1(x_0)$) recourse functions.

4.3.3 Convergence

The first question that arises with sampled models concerns how large N needs to be. Fortunately, the sample average approximation enjoys some nice convergence properties. We start by defining

$$\begin{aligned} F(x) &= \mathbb{E}F(x,W), \\ \overline{F}^N(x) &= \frac{1}{N}\sum_{n=1}^{N} F(x,W^n). \end{aligned}$$

The simplest (and most intuitive) result is that we get closer to the optimal solution as the sample size grows. We write this by saying

$$\lim_{N\to\infty} \overline{F}^N(x) \to \mathbb{E}F(x,W).$$

Let x^N be the optimal solution of the approximate function, which is to say

$$x^N = \arg\max_{x\in\mathcal{X}} \overline{F}^N(x).$$

The asymptotic convergence means that we will eventually achieve the optimum solution, a result we state by writing

$$\lim_{N\to\infty} \overline{F}^N(x^N) \to F(x^*).$$

These results tell us that we will eventually achieve the best possible objective function (note that there may be more than one optimal solution). The most interesting and important result is the rate at which we achieve this result. We start by assuming that our feasible region $\mathcal{X}$ is a set of discrete alternatives $x_1,\ldots,x_M$. This might be a set of discrete choices (e.g. different product configurations or different drug cocktails), or a discretized continuous parameter such as a price or concentration. Or, it could be a random sample of a large set of possibly vector-valued decisions.

Now, let ϵ be some small value (whatever that means). The amazing result is that as N increases, the probability that the optimal solution to the approximate problem, X^N, is more than ϵ from the optimal shrinks at an *exponential rate*. We can write this statement mathematically as

$$\mathbb{P}[F(x^N) < F(x^*) - \epsilon] < |\mathcal{X}|e^{-\eta N}, \tag{4.27}$$

for some constant $\eta > 0$. What equation (4.27) is saying is that the probability that the quality of our estimated solution x^N, given by $F(x^N)$, is more than ϵ away from the optimal $F(x^*)$, decreases at an exponential rate $e^{-\eta N}$ with a constant, $|\mathcal{X}|$, that depends on the size of the feasible region. The coefficient $\mathcal{X}$ is quite large, of course, and we have no idea of the magnitude of η. However, the result suggests that the probability that we do worse than $F(x^*) - \epsilon$ (remember that we are maximizing) declines exponentially with the same size N, which is comforting.

A similar but stronger result is available when x is continuous and $f(x, W)$ is convex, and the feasible region $\mathcal{X}$ might be specified by a set of linear inequalities. In this case, the convergence is given by

$$\mathbb{P}[F(x^N) < F(x^*) - \epsilon] < Ce^{-\eta N}, \tag{4.28}$$

for given constants $C > 0$ and $\eta > 0$. Note that unlike (4.27), equation (4.28) does not depend on the size of the feasible region, although the practical effect of this property is unclear.

The convergence rate results (4.27) (for discrete decisions) or (4.28) (for convex functions) tell us that as we allow our sample size N to increase, the optimal objective function $F(x^N)$ approaches the optimal solution $F(x^*)$ at an exponential rate, which is a very encouraging result. Of course, we never know the parameters η, or C and β, so we have to depend on empirical testing to get a sense of the actual convergence rate.

The exponential convergence rates are encouraging, but there are problems such as linear (or especially integer) programs that are computationally challenging even when $N = 1$. We are going to see these later in the context of models where we use sampling to look into the future. There are two computational issues that will need to be addressed:

Sampling - Rather than just doing random sampling to obtain $W^1, \ldots, W^N$, it is possible to choose these samples more carefully so that a smaller sample can be used to produce a more realistic representation of the underlying sources of uncertainty.

Decomposition - The sampled problem (4.15) can still be quite large (it is N times bigger than the problem we would obtain if we just used expectations for uncertain quantities), but the sampled problem has structure we can exploit using decomposition algorithms.

We defer until chapter 10 a more complete description of sampling methods to represent uncertainty. We then wait until chapter 20 to show how decomposition methods can be used in the setting of lookahead policies.

4.3.4 Decomposition strategies

From time to time, we encounter problems where the deterministic problem

$$\max_{x \in \mathcal{X}} F(x, \overline{W}),$$

where $\overline{W}$ is a point estimate of the random variable W, is reasonably difficult to solve. For example, it might be a reasonably large integer program such as might arise when scheduling airlines or planning when energy generators should turn on and off. In this case, $F(x, \overline{W})$ would be the contribution function and $\mathcal{X}$ would contain all the constraints, including integrality. Imagine that we can solve the deterministic problem, but it might not be that easy (integer programs might have 100,000 integer variables).

If we want to capture the uncertainty of W using a sample of, say, 20 different values of W, then we create an integer program that is 20 times larger. Even modern solvers on today's computers have difficulty with this. Imagine that we decompose the problem so that there is a different solution for each possible value of W? We are going to introduce notation that we will use throughout the volume by letting $\omega^1, \omega^2, \ldots, \omega^N$ be the set of outcomes of W where $W^n = W(\omega^n)$.

Now, imagine that we are going to create a solution $x(\omega)$ for each outcome. We might start by rewriting our sampled stochastic optimization problem (4.15) as

$$\max_{x(\omega^1),\ldots,x(\omega^N)} \frac{1}{N} \sum_{n=1}^{N} F(x(\omega^n), W(\omega^n)). \tag{4.29}$$

We can solve this problem by creating N parallel problems and obtaining a different solution $x(\omega^n)$ for each ω. This is like peeking into the future and then choosing the decision you would make after you see the uncertainty. This is like allowing the aircraft to arrive late to an airport because we already knew that it was going to be delayed on its next leg.

The good news is that this is a starting point. What we really want is a solution where all the $x(\omega)$ are the same. We can introduce a constraint, often known as a *nonanticipativity constraint*, that looks like

$$x(\omega^n) - \bar{x} \quad = \quad 0, \;\; n = 1, \ldots, N.$$

If we introduce this constraint, we are just back to our original (and very large) problem. But what if we relax this constraint and add it to the objective function. For example, we might solve

$$\max_{x(\omega^1),\ldots,x(\omega^N)} \frac{1}{N} \sum_{n=1}^{N} \big(F(x(\omega^n), W(\omega^n)) + \lambda^n (x(\omega^n) - \bar{x})\big). \tag{4.30}$$

What is nice about this new objective function is that, just as with the problem in (4.29), it decomposes into N problems, which makes the overall problem solvable. Now the difficulty is that we have to coordinate the different subproblems by manipulating the vector $\lambda^1, \ldots, \lambda^N$. We are not going to address this problem in detail, but this hints at a path for solving large scale problems using sampled means.

4.3.5 Creating a sampled model

A particularly important problem with large-scale applications is the design of the sample $W^1, \ldots, W^N$. The most popular methods for generating a sample are:

- From history - We may not have a formal probability model for W, but we can draw samples from history. For example, W^n might be a sample of wind speeds over a week, or currency fluctuations over a year.

- Monte Carlo simulation - There is a powerful set of tools on the computer known as Monte Carlo simulation which allow us to create samples of random variables as long as we know the underlying distribution (we cover this in more detail in chapter 10).

In some instances we have an interest in creating a reasonable representation of the underlying uncertainty with the smallest possible sample. For example, imagine that we are replacing the original problem $max_x \mathbb{E}F(x, W)$ with a sampled representation

$$\max_x \frac{1}{N} \sum_{n=1}^{N} F(x, W^n).$$

Now imagine that x is a (possibly large) vector of integer variables, which might arise if we are trying to schedule aircraft for an airline, or to design the location of warehouses for a large logistics network. In such settings, even a deterministic version of the problem might be challenging, whereas we are now trying to solve a problem that is N times as large. Instead of solving the problem over an entire sample $W^1, \ldots, W^N$, we may be interested in using a good representative subset $(W^j),\ j \in \mathcal{J}$. Assume that W^n is a vector with elements $W^n = (W_1^n, \ldots, W_k^n, \ldots, W_K^n)$. One way to compute such a subset is to compute a distance metric $d^1(n, n')$ between W^n and $W^{n'}$ which we might do using

$$d^1(n, n') = \sum_{k=1}^{K} |W_k^n - W^{n'_k}|.$$

This would be called an "L_1-norm" because it is measuring distances by the absolute value of the distances between each of the elements. We could also use an "L_2-norm" by computing

$$d^2(n, n') = \left(\sum_{k=1}^{K} (W_k^n - W^{n'_k})^2 \right)^{\frac{1}{2}}.$$

The L_2-norm puts more weight on large deviations in an individual element, rather than a number of small deviations spread over many dimensions. We can generalize this metric using

$$d^p(n, n') = \left(\sum_{k=1}^{K} (W_k^n - W^{n'_k})^p \right)^{\frac{1}{p}}.$$

However, other than the L_1 and L_2 metrics, the only other metric that is normally interesting is the L_∞-norm, which is the same as setting $d^\infty(n, n')$ equal to the absolute value of the largest difference across all the dimensions.

Using the distance metric $d^p(n, n')$, we choose a number of clusters J and then organize the original set of observations $W^1, \ldots, W^n$ into J clusters. This can be done using a popular family of algorithms that go under names such as k-means clustering or k-=nearest neighbor clustering. There are different variations of these algorithms which can be found in standard libraries such as R and Matlab. The core idea in these procedures can be roughly described as:

Step 0 Use some rule to pick J centroids. This might be suggested by problem structure, or you can pick J elements out of the set $W^1, \ldots, W^N$ at random.

Step 1 Now step through each $W^1, \ldots, W^N$ and assign each one to the centroid that minimizes the distance $d^p(n, j)$ over all centroids $j \in \mathcal{J}$.

Step 2 Find the centroids of each of the clusters and return to Step 1 until you find that your clusters are the same as the previous iteration (or you hit some limit).

This simple process is easy to code, but we recommend using one of the standard routines in libraries such as R or Matlab if these are available. A nice feature of this approach is that it can be applied to even high-dimensional random variables W, as might arise when W represents observations (wind speed, prices) over many time periods, or if it represents observations of the attributes of groups of people (such as medical patients).

The challenge of representing uncertain events using well-designed samples is growing into a mature literature. We refer the reader to the bibliographic notes for some guidance as of the time that this volume is being written.

4.4 ADAPTIVE LEARNING ALGORITHMS

When we cannot calculate the expectation exactly, either through structure or resorting to a sampled model, we have to resort to adaptive learning algorithms. This transition fundamentally changes how we approach stochastic optimization problems, since any adaptive algorithm can be modeled as a sequential decision problem, otherwise known as a dynamic program.

We separate our discussion of adaptive learning algorithms between derivative-based algorithms, discussed in chapter 5, and derivative-free algorithms, presented in chapter 7. In between, chapter 6 discusses the problem of adaptively learning a signal, a problem that introduces the annoying but persistent problem of stepsizes that we first encounter in chapter 5, but which pervades the design of adaptive learning algorithms.

We begin by offering a general model of adaptive learning problems, which are basically a simpler example of the dynamic programs that we consider later in the book. As we illustrate in chapters 5 and 7, adaptive learning methods can be viewed as sequential decision problems (dynamic programs) where the state variable captures only what we know about the problem. This gives us an opportunity to introduce some of the core ideas of sequential decision problems, without all the richness and complexity that come with this problem class.

Below, we sketch the core elements of any sequential decision problem, and then outline the fundamental class of policies (sometimes called algorithms) that are used to solve them.

4.4.1 Modeling adaptive learning problems

Whether we are solving a derivative-based or derivative-free problem, any adaptive learning algorithm is going to have the structure of a sequential decision problem, which has five core components:

State S^n - This will capture the current point in the search, and other information required by the algorithm. The nature of the state variable depends heavily on how we are structuring our search process. For now, the state variable will only capture the belief about a function. In chapter 9, we tackle the problem of modeling general dynamic programs which include states that are directly controllable (most often, these are physical problems).

Decision - While this is sometimes x^n, the precise "decision" being made within an adaptive learning algorithm depends on the nature of the algorithm, as we see in chapter 5. Depending on the setting, decisions are made by a decision rule, an algorithm, or (the term we primarily use), a policy. If x is our decision, we designate $X^\pi(S)$ as the policy (or algorithm).

Exogenous information W^n - This is the new information that is sampled during the nth iteration, either from a Monte Carlo simulation or observations from an exogenous process (which could be a computer simulation, or the real world).

Transition function - The transition function includes the equations that govern the evolution from S^n to S^{n+1}. Our default notation used throughout this volume is to write

$$S^{n+1} = S^M(S^n, x^n, W^{n+1}).$$

Objective function - This is how we evaluate how well the policy is performing. The notation depends on the setting. We may have a problem where we make a decision x^n at the end of iteration n, then observe information W^{n+1} in iteration $n+1$, from which we can evaluate our performance using $F(x^n, W^{n+1})$. This is going to be our default notation for learning problems. When we make the transition to more complex problems with a physical state, we are going to encounter problems where the contribution (cost if minimizing) depends on the state S^n and decision x^n, which we would write as $C(S^n, x^n)$. The contribution might also depend on W^{n+1}, in which case we would write it as $C(S^n, x^n, W^{n+1})$, or it might depend on the downstream state S^{n+1}, in which case we would write it as $C(S^n, x^n, S^{n+1})$, a form that is popular when the transition function is unknown. We return to the objective function below.

We are going to be able to model any sequential learning algorithm as a sequential decision process that can be modeled as the sequence

$$(S^0, x^0 = X^\pi(S^0), W^1, S^1, x^1 = X^\pi(S^1), W^2, \ldots).$$

Thus, all sequential learning algorithms, for *any* stochastic optimization problem, can ultimately be reduced to a sequential decision problem, otherwise known as a dynamic program.

For now (which is to say, chapters 5 and 7), we are going to limit our attention to where decisions only affect what we learn about the function we are optimizing. In chapter 8, we are going to introduce the complex dimension of controllable physical states. Mathematically, there is no difference in how we formulate a problem where the state consists only of what we know about a function, versus problems where the state captures the locations of people, equipment and inventory. However, pure learning problems are much simpler, and represent a good starting point for modeling and solving stochastic optimization problems using sequential (adaptive) methods.

4.4.2 Online vs. offline applications

The terms "online" and "offline" are terms that are widely used in both machine learning and stochastic optimization settings, but they take on different interpretations which can be quite important, and which have created considerable confusion in the literature. Below we explain the terms in the context of these two communities, and then describe how these terms are used in this volume.

Machine learning

Machine learning is an optimization problem that involves minimizing the error between a proposed model (typically parametric) and a dataset. We can represent the model by $f(x|\theta)$ where the model may be linear or nonlinear in θ (see chapter 3). The most traditional representation is to assume that we have a set of input variables $x_1, \ldots, x_n$ with a corresponding set of observations $y_1, \ldots, y_n$, to which we are going to fit our model by solving

$$\min_{\theta} \sum_{i=1}^{n} (y_i - f(x_i|\theta))^2, \tag{4.31}$$

where we might represent the optimal solution to (4.31) by θ^*. This problem is solved as a batch optimization problem using any of a set of deterministic optimization algorithms. This process is classically known as offline learning in the machine learning. Once we find θ^*, we would presumably use our model $f(x|\theta^*)$ to make forecasts in the future. Of course, we are hoping that our model fitted using the result of solving equation (4.31) minimizes the error between the model $f(x|\theta^*)$ and the observations $y^1, \ldots, y^n$, but these errors are not counted in our initial fitting.

In online learning, we assume that data is arriving sequentially over time. In this case, we are going to assume that we see x^n and then observe y^{n+1}, where the use of $n+1$ is our way of showing that y^{n+1} is observed after seeing $x^0, \ldots, x^n$. Let D^n be our dataset at time n where

$$D^n = \{x^0, y^1, x^1, y^2, \ldots, x^{n-1}, y^n\}.$$

We need to estimate a new value of θ, which we call θ^n, for each new piece of information which includes (x^{n-1}, y^n). We would call any method we use to compute θ^n a *learning policy*, but one obvious example would be

$$\theta^n = \arg\min_{\theta} \sum_{i=0}^{n-1} (y^{i+1} - f(x^i|\theta))^2.$$

More generally, we could write our learning policy as $\theta^n = \Theta^\pi(D^n)$. As our dataset evolves $D^1, D^2, \ldots, D^n, D^{n+1}, \ldots$, we update our estimate θ^n sequentially.

In the eyes of the machine learning community, the difference between the offline problem in equation (4.31) and the online learning problem in (4.32) is that the first is a single, batch optimization problem, while the second is implemented sequentially.

Optimization

Imagine that we are trying to design a new material to maximize the conversion of solar energy to electricity. We will go through a series of experiments testing different materials, as well as continuous parameters such as the thickness of a layer of a material. We wish to sequence our experiments to try to create a surface that maximizes energy conversion within our experimental budget. What we care about is how well we do in the end; trying a design that does not work is not a problem as long as the final design works well.

Now consider the problem of actively tilting solar panels to maximize the energy production over the course of the day, where we have to handle not just the changing angle of the sun during the day (and over seasons), but also with changes in cloud cover. Again, we may have to experiment with different angles, but now we need to maximize the total energy created while we are trying to learn the best angle.

We would treat the first problem as an offline learning problem since we are learning in the lab. Because we are in the lab, we do not mind failed experiments as long as we get the best result in the end. Let x^n specifies our experimental choices after we have run n experiments (types of materials, thicknesses) and let W^{n+1} is the power generated by the material from the $n+1st$ experiment. Finally assume we are choosing x^n using some experimental policy $X^\pi(S^n)$ where S^n captures our belief state after n experiments. If we run N experiments, let $x^{\pi,N}$ be the final design given what we know (captured by S^N) after our budget of N experiments has been. We evaluate the performance of the policy using

$$F^\pi = \mathbb{E}F(x^{\pi,N}, W),$$

where the expectation means testing the design over different conditions. A nice goal is to find the policy that produces the best performance, which we might write as

$$\max_\pi F^\pi.$$

We call this the "final reward."

In the second problem, where x is now the tilt in the solar panel, we need to find a control strategy (we will call it a policy) $X^\pi(S_t)$ given what we know at time t. Let $x_t = X^\pi(S_t)$ be the control used during the period t to $t+1$ after which we observe the energy generated given by W_{t+1}. We want to pick a policy that solves

$$\max_\pi \mathbb{E} \sum_{t=0}^{T} F(X^\pi(S_t), W_{t+1}). \tag{4.32}$$

We call this the "cumulative reward."

Perspectives of offline and online

These two examples illustrate two perspectives of offline vs. online. In the machine learning community, "offline" means batch, while "online" means sequential, and it does not matter if the algorithm is being run sequentially "offline" on data to estimate a model to be used later, or if it is being used in the field where data is arriving over time (which forces sequential updates).

In optimization problems (and yes, we know that machine learning is an optimization problem) offline means that we need to find a design where all that matters is the final design, and not how we got there. Further, while optimization can be done in batch (for example, by solving a linear programming problem using a solver), there are many problems (such as the experimental setting we described above) where offline learning might be done using a sequential process (but in the lab). By contrast, if the optimization is being run in the field, then we have to live with our performance over time. Thus, an optimization algorithm can be fully sequential but run in the lab where we only care about the performance of the final design $x^{\pi,N}$, or in the field where we have to live with the results of each experiment.

In this volume, we will try to use the terms "offline" and "online" in a way that is sensitive to both perspectives. In particular, we will make the distinction between "final reward" (or cost), and "cumulative reward" (or cost), where final reward refers to the final design (and not how we got there), while cumulative reward means we are adding up our performance over the learning iterations.

4.4.3 Objective functions for learning

In contrast with the exact methods for solving stochastic optimization problems, there are different ways to formulate the objective function for adaptive learning problems. For learning problems, we are going to let $F(x, W)$ be the function that captures our performance objective when we make decision x and then observe random information W. In an iterative setting, we will write $F(x^n, W^{n+1})$; in a temporal setting, we will write $F(x_t, W_{t+1})$. Our choice $x^n = X^\pi(S^n)$ will be made by a policy that depends on the state, but otherwise the contribution $F(x, W)$ depends only on the action and random information.

The function $\mathbb{E}F(x, W)$ captures the performance of our implementation decision x. To make a good decision, we need to design an algorithm, or more precisely, a learning policy $X^\pi(S)$, that allows us to find the best x. There are different objective functions for capturing the performance of a learning policy:

Terminal reward - Let $x^{\pi,n} = X^\pi(S^n)$ be our solution at iteration n while following policy π. We may analyze the policy π in two ways:

Finite time analysis - Here, we want to solve

$$\max_\pi \mathbb{E}F(x^{\pi,N}, W) = \mathbb{E}_{W^1,\ldots,W^N}\mathbb{E}_{x^{\pi,N}|W^1,\ldots,W^N}\mathbb{E}_W F(x^{\pi,N}, W). \quad (4.33)$$

Asymptotic analysis - In this setting, we are trying to establish that

$$\lim_{N\to\infty} x^{\pi,N} \to x^*$$

where x^* solves $\max_x \mathbb{E}F(x, W)$. In both of these settings, we are only interested in the quality of the final solution, whether it is $x^{\pi,N}$ or x^*. We do not care about the solutions obtained along the way.

Cumulative reward - Cumulative reward objectives arise when we are interested not just in the performance after we have finished learning the best asymptotic design x^*, or the best design in a finite budget N, $x^{\pi,N}$, or finite time T, x_T^π. We divide these problems into two broad classes:

Deterministic policy - The most common setting is where we want to design a single policy that optimizes the cumulative reward over some horizon. We can further divide static policies into two classes:

Stationary policy - This is the simplest setting, where we wish to find a single policy $X^\pi(S_t)$ to solve:

$$\max_\pi \mathbb{E}\sum_{t=0}^{T-1} F(X^\pi(S_t), W_{t+1}). \quad (4.34)$$

within a finite time horizon T. We may write this in both discounted and average reward forms:

$$\max_\pi \mathbb{E}\sum_{t=0}^{T} \gamma^t C(S_t, X^\pi(S_t)). \quad (4.35)$$

$$\max_\pi \mathbb{E}\frac{1}{T}\sum_{t=0}^{T} C(S_t, X^\pi(S_t)). \quad (4.36)$$

Both (4.35) and (4.36) can be extended to infinite horizon, where we would replace (4.36) with

$$\max_{\pi} \lim_{T \to \infty} \mathbb{E} \frac{1}{T} \sum_{t=0}^{T} C(S_t, X^\pi(S_t)). \tag{4.37}$$

Nonstationary policy - There are many problems where we need a time-dependent policy $X_t^\pi(S_t)$, either because the behavior needs to vary by time of day, or because we need different behaviors based on how close the decisions are to the end of horizon. We denote the policy by time t as $X_t^\pi(S_t)$, but let π_t refer to the choices (type of function, parameters) we need to make for each time period. These problems would be formulated

$$\max_{\pi_0, \ldots, \pi_{T-1}} \mathbb{E} \sum_{t=0}^{T-1} F(X_t^\pi(S_t), W_{t+1}). \tag{4.38}$$

Although the policies are time dependent, they are in the class of static policies because they are designed before we start the process of making observations.

Adaptive policy - Now we allow our policies to learn over time, as would often happen in an online setting. Modeling this is a bit subtle, and it helps to use an example. Imagine that our policy is of the form

$$X^\pi(S_t|\theta) = \theta_0 + \theta_1 S_t + \theta_2 S_t^2.$$

This would be an example of a stationary policy parameterized by $\theta = (\theta_0, \theta_1, \theta_2)$. Now imagine that θ is a function of time, so we would write our policy as

$$X^\pi(S_t|\theta_t) = \theta_{t0} + \theta_{t1} S_t + \theta_{t2} S_t^2.$$

Finally, imagine that we have an adaptive policy that updates θ_t after computing $x_t = X^\pi(S_t|\theta_t)$ and observing W_{t+1}. Just as we have to make a decision x_t, we have to "decide" on how to set θ_{t+1} given S_{t+1} (which depends on S_t, x_t and W_{t+1}). Imagine that we call the θ-updating policy Θ^π, where we would write

$$\theta_t = \Theta^{\pi^\theta}(S_t).$$

We refer to $\Theta^{\pi^\theta}(S_t)$ as the *learning policy* while $X^\pi(S_t|\theta_t)$ as the *implementation policy*. This problem is formulated as

$$\max_{\pi} \max_{\pi^\theta} \mathbb{E} \sum_{t=0}^{T-1} F(X^\pi(S_t|\theta_t), W_{t+1}).$$

where π refers to the design of the implementation policy $X^\pi(S_t|\theta_t)$ while $\theta_t = \Theta^{\pi^\theta}(S_t)$ is the policy that updates any parameters in the implementation policy (technically, this policy depends on the structure of the implementation policy π). As always, $S_{t+1} = S^M(S_t, x_t = X^\pi(S_t|\theta_t), W_{t+1})$. We leave to later the description of how to model and search over learning policies.

For learning problems (problems where the *function* $F(x, W)$ does not depend on the state), we are going to use (4.33) (the terminal reward) or (4.34) (the cumulative reward for stationary policies) as our default notation for the objective function.

It is common, especially in the machine learning community, to focus on *regret* rather than the total reward, cost or contribution. Regret is simply a measure of how well you do relative to how well you could have done (but recognize that there are different ways of defining the best we could have done). For example, imagine that our learning policy has produced the approximation $\overline{F}^{\pi,N}(x)$ of the function $\mathbb{E}F(x, W)$ by following policy π after N samples, and let

$$x^{\pi,N} = \arg\max_x \overline{F}^{\pi,N}(x)$$

be the best solution based on the approximation. The regret $\mathcal{R}^{\pi,N}$ would be given by

$$\mathcal{R}^{\pi,N} = \max_x \mathbb{E}F(x, W) - \mathbb{E}F(x^{\pi,N}, W). \tag{4.39}$$

Of course, we cannot compute the regret in a practical application, but we can study the performance of algorithms in a setting where we assume we know the true function (that is, $\mathbb{E}F(x, W)$), and then compare policies to try to discover this true value. Regret is popular in theoretical research (for example, computing bounds on the performance of policies), but it can also be used in computer simulations comparing the performance of different policies.

4.4.4 Designing policies

Now that we have presented a framework for modeling our learning problems, we need to address the problem of designing policies (we will sometimes refer to these as algorithms), especially in chapter 5 when we deal with derivative-based optimization.

There are two fundamental strategies for designing policies, each of which break down into two subclasses, creating four classes of policies:

Policy search - These are functions that are tuned to work well over time without directly modeling the effect of a decision now on the future. Policies designed using policy search fall into two styles:

Policy function approximations (PFAs) - PFAs are analytical functions that map directly from state to action.

Cost function approximations (CFAs) - CFAs involve maximizing (or minimizing) a parametric function.

Lookahead policies - These are policies that are designed by estimating, directly or indirectly, the impact of a decision now on the future. There are again two ways of creating these policies:

Value function approximations (VFAs) - If we are in a state S^n, take an action x^n, that leads (with the introduction of new information) to a new state S^{n+1}, assume we have a function $V^{n+1}(S^{n+1})$ that estimates (exactly or, more often, approximately) the value of being in state S^{n+1}. The value function $V^{n+1}(S^{n+1})$ captures the impact of decision x^n.

Direct lookahead policies (DLAs) - These are policies where we model the downstream trajectory of each decision before deciding which decision to make now.

The importance of each of these four classes depends on the characteristics of the problem. We are going to see all four of these classes used in the setting of derivative-free optimization in chapter 7. By contrast, derivative-based search strategies reviewed in chapter 5 have historically been more limited, although this perspective potentially introduces new strategies that might be pursued. When we transition to problems with physical states starting in chapter 8, we are going to see that we will need to draw on all four classes. For this reason, we discuss these four classes in more depth in chapter 11.

4.5 CLOSING REMARKS

This chapter offers three fundamental perspectives of stochastic optimization problems. Section 4.2 is basically a reminder that any stochastic optimization problem can be solved as a deterministic optimization problem if we are able to compute the expectation exactly. While this will not happen very often, we offer this section as a reminder to readers not to overlook this path.

Section 4.3 then introduces the powerful approach of using sampled models, where we overcome the complexity of computing an expectation by replacing the underlying uncertainty model with a small sampled set, which is much easier to model. This strategy should always be in your toolbox, even when it will not solve the entire problem.

When all else fails (which is most of the time), we are going to need to turn to adaptive learning strategies, which we introduce in section 4.4. These approaches have evolved into substantial areas of research, which we divide into derivative-based methods in chapter 5, and derivative-free methods in chapter 7. In chapter 5, we are going to see that we need a device called "stepsizes" (which we cover in chapter 6), which can be viewed as a type of decision, where different stepsize rules are actually types of policies.

4.6 BIBLIOGRAPHIC NOTES

- Section xx - A good reference for contextual bandit problems is given in Bubeck & Cesa-Bianchi (2012).

PROBLEMS

4.1 In a flexible spending account (FSA), a family is allowed to allocate x pretax dollars to an escrow account maintained by the employer. These funds can be used for medical expenses in the following year. Funds remaining in the account at the end of the following year are given back to the employer. Assume that you are in a 35 percent tax bracket (sounds nice, and the arithmetic is a bit easier).

Let W be the random variable representing total medical expenses in the upcoming year, and let $P^W(S) = Prob[W \leq w]$ be the cumulative distribution function of the random variable W.

a) Write out the objective function $F(x)$ that we would want to solve to find x to minimize the total cost (in pretax dollars) of covering your medical expenses next year.

b) If x^* is the optimal solution and $\nabla_x F(x)$ is the gradient of your objective function if you allocate x to the FSA, use the property that $\nabla_x F(x) = 0$ to derive the critical ratio that gives the relationship between $x*$ and the cumulative distribution function $P^W(w)$.

c) Given your 35 percent tax bracket, what percentage of the time should you have funds left over at the end of the year?

4.2 Consider the problem faced by a mutual fund manager who has to decide how much to keep in liquid assets versus investing to receive market returns. Assume he has R_t dollars to invest at the end of day t, and needs to determine the quantity x_t to put in cash at the end of day t to meet the demand $\hat{D}_{t+1}$ for cash in day $t+1$. The remainder, $R_t - x_t$, is to be invested and will receive a market return of $\hat{\rho}_{t+1}$ (for example, we might have $\hat{\rho}_{t+1} = 1.0002$, implying a dollar invested is worth 1.0002 tomorrow). Assume there is nothing earned for the amount held in cash.

If $\hat{D}_t > x_{t-1}$, the fund manager has to redeem stocks. Not only is there a transaction cost of 0.20 percent (redeeming $1000 costs $2.00), the manager also has to pay capital gains. His fund pays taxes on the average gain of the total assets he is holding (rather than the gain on the money that was just invested). At the moment, selling assets generates a tax commitment of 10 percent which is deducted and held in escrow. Thus, selling $1000 produces net proceeds of 0.9(1000 â^ 2). As a result, if he needs to cover a cash request of $10,000, he will need to sell enough assets to cover both the transaction costs (which are tax deductible) and the taxes, leaving $10,000 net proceeds to cover the cash request.

a) Formulate the problem of determining the amount of money to hold in cash as a stochastic optimization problem. Formulate the objective function $F(x)$ giving the expected return when holding x dollars in cash.

b) Give an expression for the stochastic gradient $\nabla_x F(x)$.

c) Find the optimal fraction of the time that you have to liquidate assets to cover cash redemption. For example, if you manage the fund for 100 days, how many days would you expect to liquidate assets to cover cash redemptions?

4.3 Independent system operators (ISOs) are companies that manage our power grid by matching generators (which create the energy) with customers. Electricity can be generated via steam, which takes time, or gas turbines which are fast but expensive. Steam generation has to be committed in the day-ahead market, while gas turbines can be brought on line with very little advance notification.

Let x_t be the amount of steam generation capacity (measured in megawatt-hours) that is requested on day t to be available on day $t+1$. Let $p^{steam}_{t,t+1}$ be the price of steam on day $t+1$ that is bid on day t (which is known on day t). Let D_{t+1} be the demand for electricity (also measured in megawatt-hours) on day $t+1$, which depends on temperature and other factors that cannot be perfectly forecasted. However, we do know the cumulative distribution function of D_{t+1} , given by $F^D(d) = Prob[D_{t+1} < d]$. If the demand exceeds the energy available from steam (planned on day t), then the balance has to be generated from gas turbines. These are bid at the last minute, and therefore we have to pay a random price p^{GT}_{t+1}. At the same time, we are not able to store energy; there is no inventory held over if $D_{t+1} < x_t$. Assume that the demand D_{t+1} and the price of electricity from gas turbines p^{GT}_{t+1} are independent.

a) Formulate the objective function $F(x)$ to determine x_t as an optimization problem.

b) Compute the stochastic gradient of your objective function $F(x)$ with respect to x_t. Identify which variables are known at time t, and which only become known at time $t+1$.

c) Find an expression that characterizes the optimal value of x_t in terms of the cumulative probability distribution $F^D(d)$ of the demand D_T.

4.4 We are going to illustrate the difference between

$$\max_x \mathbb{E}F(x, W) \tag{4.40}$$

and

$$\max_x F(x, \mathbb{E}W) \tag{4.41}$$

using a sampled belief model. Assume we are trying to price a product where the demand function is given by

$$D(p|\theta) = \theta^0 \frac{e^{U(p|\theta)}}{1 + e^{U(p|\theta)}}, \tag{4.42}$$

where

$$U(p|\theta) = \theta_1 + \theta_2 p.$$

Our goal is to find the price that maximizes total revenue given by

$$R(p|\theta) = pD(p|\theta). \tag{4.43}$$

Here, our random variable W is the vector of coefficients $\theta = (\theta_0, \theta_1, \theta_2)$ which can take one of four possible values of θ given by the set $\Theta = \{\theta^1, \theta^2, \theta^3, \theta^4\}$.

θ	$P(\theta)$	θ_0	θ_1	θ_2
θ^1	0.20	50	4	-0.2
θ^2	0.35	65	4	-0.3
θ^3	0.30	75	4	-0.4
θ^4	0.15	35	7	-0.25

Table 4.1 Data for exercise 4.4.

a) Find the price $p(\theta)$ that maximizes

$$\max_p R(p|\theta), \tag{4.44}$$

for each of the four values of θ. You may do this analytically, or to the nearest integer (the relevant range of prices is between 0 and 40). Either way, it is a good idea to plot the curves (they are carefully chosen). Let $p^*(\theta)$ be the optimal price for each value of θ and compute

$$R^1 = \mathbb{E}_\theta \max_{p(\theta)} R(p^*(\theta)|\theta). \tag{4.45}$$

b) Find the price p that maximizes

$$R^2 = \max_p \mathbb{E}_\theta R(p|\theta), \tag{4.46}$$

where $R(p|\theta)$ is given by equation (4.43).

c) Now find the price p that maximizes

$$R^3 = \max_p R(p|\mathbb{E}\theta).$$

d) Compare the optimal prices and the optimal objective functions R^1, R^2 and R^3 produced by solving (4.44), (4.46) and (4.47). Use the relationships among the revenue functions to explain as much as possible about the relevant revenues and prices.

4.5

CHAPTER 5

DERIVATIVE-BASED STOCHASTIC SEARCH

We begin our discussion of adaptive learning methods in stochastic optimization by addressing problems where we have access to derivatives (or gradients, if x is a vector) of our function $F(x, W)$. We are going to focus primarily on the asymptotic form of our basic stochastic optimization problem

$$\max_{x \in \mathcal{X}} \mathbb{E}\{F(x, W)|S_0\}, \tag{5.1}$$

but later we are going to shift attention to finding the best algorithm (or policy) for finding the best solution within a finite budget. We are going to show that with any adaptive learning algorithm, we can define a state S^n that captures what we know after n iterations, and we can represent any algorithm as a "policy" π, which produces a solution $x^{\pi,N}$ after we exhaust our budget of N iterations. When we focus on this finite-budget setting, the problem in (5.1) becomes

$$\max_{\pi} \mathbb{E}\{F(X^{\pi,N}, W)|S_0\}. \tag{5.2}$$

The transition from searching for a real-valued vector x to finding a function π is one of the central differences between deterministic optimization and most stochastic optimization problems, which is to say all stochastic optimization problems that involve adaptive learning.

In this chapter, we assume that we can compute the gradient $\nabla F(x, W)$ once the random information W becomes known. This is most easily illustrated using the newsvendor problem. Let x be the number of newspapers placed in a bin, with unit cost c. Let W be the random demand for newspapers (which we learn after choosing x), which are sold at price p. We wish to find x that solves

$$\max_x F(x) = \mathbb{E}F(x,W) = \mathbb{E}\big(p\min\{x,W\} - cx\big). \tag{5.3}$$

We can use the fact that we can compute *stochastic gradients*, which are gradients that we compute only after we observe the demand W, given by

$$\nabla_x F(x,W) = \begin{cases} p-c & \text{If } x \leq W \\ -c & \text{If } x > W. \end{cases} \tag{5.4}$$

We are going to show how to design simple algorithms that exploit our ability to compute gradients after the random information becomes known. Even when we do not have direct access to gradients, we may be able to estimate them using finite differences. We are also going to see that the core ideas of stochastic gradient methods pervade a wide range of adaptive learning algorithms.

We start by summarizing a variety of applications.

5.1 SOME SAMPLE APPLICATIONS

Derivative-based problems exploit our ability to use the derivative after the random information has been observed (but remember that our decision x must be made before we have observed this information). These derivatives, known as stochastic gradients, require that we understand the underlying dynamics of the problem. When this is available, we have access to some powerful algorithmic strategies that have been developed since these ideas where first invented in 1951 by Robbins and Monro.

Some examples of problems where derivatives can be computed directly are:

- Cost minimizing newsvendor problem - A different way of expressing the newsvendor problem is one of minimizing overage and underage costs. Using the same notation as above, our objective function would be written

  $$\min_x \mathbb{E}F(x,W) = \mathbb{E}[c^o \max\{0, x-W\} + c^u \max\{0, W-x\}]. \tag{5.5}$$

  We can compute the derivative of $F(x,\hat{D})$ with respect to x after W becomes known using

  $$\nabla_x F(x,W) = \begin{cases} c^0 & \text{If } x > W \\ -c^u & \text{If } x \leq W. \end{cases}$$

  The gradient $\nabla_x F(x,W)$ is known as a stochastic gradient because it depends on the random demand W.

- Nested newsvendor - This hints at a multidimensional problem which would be hard to solve even if we knew the demand distribution. Here there is a single random demand D that we can satisfy with products $1,\ldots,K$ where we use the supply of products $1,\ldots,k-1$ before using product k. For example, imagine that k. The profit maximizing version is given by

  $$\max_{x_1,\ldots,x_K} = \sum_{k=1}^K p_k \mathbb{E}\min\left\{x_k, \left(D - \sum_{j=1}^{k-1} x_j\right)^+\right\} - \sum_{k=1}^K c_k x_k. \tag{5.6}$$

Although more complicated than the scalar newsvendor, it is still fairly straightforward to find the gradient with respect to the vector x once the demand becomes known.

- Statistical learning - Let $f(x|\theta)$ be a statistical model which might be of the form

  $$f(x|\theta) = \theta_0 + \theta_1\phi_1(x) + \theta_2\phi_2(x) + \dots$$

  Imagine we have a dataset of input variables $x^1, \dots, x^N$ and corresponding response variables $y^1, \dots, Y^N$. We would like to find θ to solve

  $$\min_\theta \frac{1}{N}\sum_{n=1}^{N}(y^n - f(x^n|\theta))^2.$$

- Finding the best inventory policy - Let R_t be the inventory at time t. Assume we place an order x_t according to the rule

  $$X^\pi(R_t|\theta) = \begin{cases} \theta^{max} - R_t & \text{If } R_t < \theta^{min} \\ 0 & \text{Otherwise.} \end{cases}$$

  Our inventory evolves according to

  $$R_{t+1} = \max\{0, R_t + x_t - D_{t+1}\}.$$

  Assume that we earn a contribution $C(R_t, x_t, D_{t+1})$

  $$C(R_t, x_t, D_{t+1}) = p\min\{R_t + x_t, D_{t+1}\} - cx_t.$$

  We then want to choose θ to maximize

  $$\max_\theta \mathbb{E}\sum_{t=0}^{T} C(R_t, X^\pi(R_t|\theta), D_{t+1}).$$

  If we let $F(x, W) = \sum_{t=0}^{T-1} C(R_t, X^\pi(R_t|\theta), D_{t+1})$ where $x = (\theta^{min}, \theta^{max})$ and $W = D_1, D_2, \dots, D_T$, then we have the same problem as our newsvendor problem in equation (5.3). In this setting, we simulate our policy, and then look back and determine how the results would have changed if θ is perturbed for the same sample path. It is sometimes possible to compute the derivative analytically, but if not, we can also do a numerical derivative (but using the same sequence of demands).

- Maximizing revenue on Amazon - Assume that demand for a product is given by

  $$D(p|\theta) = \theta_0 - \theta_1 p + \theta_2 p^2.$$

  Now, find the price p to maximize the revenue $R(p) = pD(p|\theta)$ where θ is unknown.

- Optimizing engineering design - An engineering team has to tune the timing of a combustion engine to maximize fuel efficiency while minimizing emissions. Assume the design parameters x include the pressure used to inject fuel, the timing of the beginning of the injection, and the length of the injection. From this the engineers observe the gas consumption $G(x)$ for a particular engine speed, and the emissions $E(x)$, which are combined into a utility function $U(x) = U(E(x), G(x))$ which combines emissions and mileage into a single metric. $U(x)$ is unknown, so the goal is to find an estimate $\bar{U}(x)$ that approximates $U(x)$, and then maximize it.

- Derivatives of simulations - In the previous section we illustrated a stochastic gradient algorithm in the context of a fairly simple, stochastic function. But imagine that we have a multiperiod simulation, such as we might encounter when simulating flows of jobs around a manufacturing center. Perhaps we use a simple rule to govern how jobs are assigned to machines once they have finished a particular step (such as being drilled or painted). However, these rules have to reflect physical constraints such as the size of buffers for holding jobs before a machine can start working on them. If the buffer for a downstream machine is full, the rule might specify that a job be routed to a different machine or to a special holding queue.

  This is an example of a policy that is governed by static variables such as the size of the buffer. We would let x be the vector of buffer sizes. It would be helpful, then, if we could do more than simply run a simulation for a fixed vector x. What if we could compute the derivative with respect to each element of x, so that after running a simulation, we obtain all the derivatives?

  Computing these derivatives from simulations is the focus of an entire branch of the simulation community. A class of algorithms called *infinitessimal perturbation analysis* was developed specifically for this purpose. It is beyond the scope of our presentation to describe these methods in any detail, but it is important for readers to be aware that the field exists.

5.2 MODELING UNCERTAINTY

Before we progress too far, we need to pause and say a few words about how we are modeling uncertainty, and the meaning of what is perhaps the most dangerous piece of notation in stochastic optimization, the expectation operator $\mathbb{E}$.

We are going to talk about uncertainty from three perspectives. The first is the random variable W that arises when we evaluate a solution, which we refer to as *implementation uncertainty*. The second is the initial state S_0, where we express *model uncertainty*, typically in the form of uncertainty about parameters (but sometimes in the structure of the model itself). Finally, we are going to discuss the uncertainty in implementing an algorithm (or policy) to find a solution, which we refer to as *algorithmic variability*.

5.2.1 Implementation uncertainty W

We illustrate implementation uncertainty using our newsvendor problem, where we make a decision x, then observe a random demand $W = \hat{D}$, after which we calculate our profit using equation (5.3). Imagine that our demand follows a Poisson distribution given by

$$\mathbb{P}[\hat{D} = d] = \frac{\mu^d e^{-\mu}}{d!},$$

where $d = 0, 1, 2, \ldots$. In this setting, our expectation would be over the possible outcomes of $\hat{D}$, so we could write (5.3) as

$$F(x|\mu) = \sum_{d=0}^{\infty} \frac{\mu^d e^{-\mu}}{d!} \big(p \min\{x, d\} - cx\big).$$

ω	$\hat{D}^1$	$\hat{D}^2$	$\hat{D}^3$	$\hat{D}^4$	$\hat{D}^5$	$\hat{D}^6$	$\hat{D}^7$	$\hat{D}^8$	$\hat{D}^9$	$\hat{D}^{10}$
1	0	1	6	3	6	1	6	0	2	4
2	3	2	2	1	7	5	4	6	5	4
3	5	2	3	2	3	4	2	7	7	5
4	6	3	7	3	2	3	4	7	3	4
5	3	1	4	5	2	4	3	4	3	1
6	3	4	4	3	3	3	2	2	6	1

Table 5.1 Illustration of six sample paths for the random variable $\hat{D}$.

It is actually quite rare that we not only have a probability distribution to describe our uncertainty, but that we can even use it. There are so many problems where randomness is multidimensional, which means that our expectation is a multidimensional sum or integral. These expectations are computationally intractable if we have more than two or three dimensions.

For these cases, we can typically assume that there is some way to generate random realizations. In our example where W is the demand $\hat{D}$, we assume that we can generate a random sequence $\hat{D}^1, \hat{D}^2, \ldots, \hat{D}^N$. We let the Greek letter ω represent an instance of a set of N sample realizations of $\hat{D}$. The set Ω is the set of every possible realization, or we might generate a sample of, say, N sample realizations, and assume that this represents our uncertainty. In this case, we would call our sampled set of outcomes $\hat{\Omega}$, where it should be the case that $\hat{\Omega}$ is a subset of the original sample space Ω. Table 5.1 illustrates six sample paths (each indexed by ω), each of length 10.

We may approximate our newsvendor problem by generating a sample of possible demands $\hat{D}$ and storing it in our set $\hat{\Omega}$. Thus, we might generate

$$\hat{\Omega} = \{1, 6, 5, 8, 2, 5, 4, 4, 3, 7\}.$$

Thus, for $\omega = 3$, $\hat{D}(\omega) = 5$.

Using this set, and assuming that each outcome is equally likely, we would write our estimate of the expectation as

$$\overline{F}(x) = \frac{1}{|\hat{\Omega}|} \sum_{\omega \in \hat{\Omega}} p \min\{x, \hat{D}(\omega)\} - cx.$$

A good exercise when looking at an equation with an expectation is to ask the question: how might you simulate an approximation of the expectation? This forces a valuable exercise, which is to think about what variables are random. Keep in mind that these random samples can come from a distribution, or an external source (the internet, a laboratory experiment, a computer simulation).

5.2.2 Model uncertainty S_0

The initial state S_0 carries both the structure of our model, but also any parameters that define how it works. It also carries probabilistic information about any uncertainties, which may be in either the structure, or the parameters.

We can illustrate uncertainty in a parameter by assuming that we do not know the mean demand μ. However, we may feel that we can describe its likely values using an exponential distribution given by

$$\mu \sim \lambda e^{-\lambda u},$$

where the parameter λ is known as a hyperparameter, which is to say it is a parameter that determines a distribution that describes the uncertainty of a problem parameter. The assumption is that even if we do not know λ precisely, it still does a good job of describing the uncertainty in the mean demand μ.

Now we have two random variables: the random demand $\hat{D}$, and the uncertain mean μ. In this case, we could write $W = (\mu, \hat{D})$. A better way would be to put the uncertainty around μ in the initial state S_0 (here it would be called a prior). In this case, we would write our problem as

$$\begin{aligned} F(x) &= \mathbb{E}\{F(x,W)|S_0\}, \\ &= \mathbb{E}_{S_0}\{\mathbb{E}_{W|S_0}F(x,W)|S_0\}. \end{aligned}$$

For our example, this would be translated as

$$F(x|\lambda) = \mathbb{E}_{\mu}\{\mathbb{E}_{\hat{D}|\mu}F(x,\hat{D})|\mu\}.$$

The notation $\mathbb{E}_{W|\mu}$ means the conditional expectation given μ. Using our distributions where the random demand W follows a Poisson distribution with mean μ which is itself random with an exponential distribution with mean λ, we would write the expectation as

$$F(x|\lambda) = \int_{u=0}^{\infty} \lambda e^{-\lambda u} \sum_{d=0}^{\infty} \frac{u^d e^{-u}}{d!}\big(p\min(x,d) - cx\big).$$

In practice, we are rarely using explicit probability distributions. One reason is that we may not know the distribution, but we may have an exogenous source for generating random outcomes. The other is that we may have a distribution, but it might be multidimensional and impossible to compute.

5.2.3 Learning uncertainty

Finally, consider an adaptive algorithm (which we first introduced in chapter 4) that proceeds by guessing x^n and then observing W^{n+1} which leads to x^{n+1} and so on (we give examples of these procedures in this chapter). If we limit the algorithm to N iterations, our sequence will look like

$$(x^0, W^1, x^1, W^2, x^2, \ldots, x^n, W^{n+1}, \ldots, x^N).$$

Below, we are going to describe the rule that produces each x^n as a policy π, so it can help to write each decision as $x^{\pi,n}$, leading to the final solution $x^{\pi,N}$.

Clearly, the final solution $x^{\pi,N}$ (as well as all the intermediate solutions) depends on the sequence $W^1, \ldots, W^N$. That means that $x^{\pi,N}$ is a random variable that depends on the sequence of observations we make while executing the algorithm. In fact, it may easily be the case that an outcome W^n depends on the previous decision x^{n-1} (or even the entire history). For this reason, we are going to designate an outcome of the sequence

$(W^1, \ldots, W^n, \ldots, W^N)$ by ω^π, which indicates that the sample path may depend on the policy. We then assume $\omega^\pi \in \Omega^\pi$, where Ω^π is the set of all possible sample paths.

When we finally obtain our solution $x^{\pi,N}$, we then have to evaluate the quality of the solution. For the moment, let's fix $x^{\pi,N}$. Now the only uncertainty is the random variable W when we go to implement $x^{\pi,N}$. However, now we are using W to test the performance of $x^{\pi,N}$. It is useful to distinguish the Ws that we use to find $x^{\pi,N}$, and the W we use to test it. For this reason, we will sometimes use $\widehat{W}$ as our "testing" W, while the sequence $W^1, \ldots, W^N$ is our "training" Ws.

We typically evaluate a policy by simulating a sequence of $W^1, \ldots, W^N$. Let $\hat{\Omega}$ be a set of possible samples (perhaps 20 of them) and let ω be one of the elements of the sample (we might think of ω as being a number from 1 to 20). Then $W^n(\omega)$ would be the actual realization of W^n for the nth sample, as illustrated in table 5.1.

Assume that we have a set of outcomes of W that we call $\hat{\Omega}$, where $\omega \in \hat{\Omega}$ is one outcome of W which we represent using $W(\omega)$. Once again assume that we have taken a random sample to create $\hat{\Omega}$ where every outcome is equally likely. Then we could evaluate our solution $x^{\pi,N}$ using

$$\overline{F}(x^{\pi,N}) = \frac{1}{|\hat{\Omega}|} \sum_{\omega \in \hat{\Omega}} F(x^{\pi,N}, W(\omega)).$$

The estimate $\overline{F}(x^{\pi,N})$ evaluates a single decision $x^{\pi,N}$, but not the policy that produced the decision. What we are really interested in is the performance of the policy, which we designate F^π. To do this, we might write

$$F^\pi = \mathbb{E}\{F(x^{\pi,N}, \widehat{W})|S_0\}. \tag{5.7}$$

The challenge is now to parse this down to something we can compute. To do this, we have to break down all the sources of uncertainty. It is instructional to unroll this in reverse as follows:

Evaluating $x^{\pi,N}$ - Given $x^{\pi,N}$, we have to simulate $\widehat{W}$ to find $F(x^{\pi,N}, \widehat{W})$.

Finding $x^{\pi,N}$ - Here we have to simulate the sequence $W^1, \ldots, W^N$ while following policy π. In some cases the policy itself is random (for example, the policy might be to choose an action x with probability $\pi(x|\theta)$).

Sampling each W^n - While following policy π, an outcome W^n depends on our model specified in S_0, which may include distributional information about a parameter (for example).

The state S_0 - If our initial state includes distributional information about a parameter, we have to sample over this distribution so that we can simulate W^n.

Recognizing this sequence, we can now rewrite our expectation in (5.7) using

$$F^\pi = \mathbb{E}_{S_0} E_{W^1,\ldots,W^N|S_0} E_{x^{\pi,N}|W^1,\ldots,W^N} E_{\widehat{W}|x^{\pi,N}} F(x^{\pi,N}, \widehat{W}).$$

In practice, we can replace each expectation by a sample over whatever is random. Furthermore, these samples can be a) sampled from a probability distribution, b) represented by a large, batch dataset, or c) observed from an exogenous process (which involves online learning).

5.2.4 Closing notes

This section is hardly a comprehensive treatment of modeling uncertainty. Given the richness of this topic, chapter 10 is dedicated to describing the process of modeling uncertainty. This chapters addresses the types of uncertainty, identifying distributions, and the process of sampling.

We mention only in passing the growing interest in replacing the expectation $\mathbb{E}$ with some form of risk measure that recognizes that the possibility of extreme outcomes is more important that is represented by their likelihood (which may be low). Expectations average over all outcomes, so if extreme events occur with low probability, they do not have much effect on the solution. Also, expectations may have the effect of letting high outcomes cancel low outcomes, when in fact one tail is much more important than the other.

5.3 STOCHASTIC GRADIENT METHODS

One of the oldest and earliest methods for solving our basic stochastic optimization problem

$$\max_x \mathbb{E}F(x, W). \tag{5.8}$$

uses the fact that we can often compute the gradient of $F(x, W)$ with respect to x after the random variable W becomes known. For example, assume that we are trying to solve a newsvendor problem, where we wish to allocate a quantity x of resources ("newspapers") before we know the demand W. The optimization problem is given by

$$\max_x F(x) = \mathbb{E}p \min\{x, W\} - cx. \tag{5.9}$$

If we could compute $F(x)$ exactly (that is, analytically), and its derivative, then we could find x^* by taking its derivative and setting it equal to zero. If this is not possible, we could still use a classical steepest ascent algorithm

$$x^n \quad = \quad x^{n-1} + \alpha_{n-1} \nabla_x F(x^{n-1}), \tag{5.10}$$

where α_{n-1} is a stepsize. For deterministic problems, we typically choose the best stepsize by solving the one-dimensional optimization problem

$$\alpha^{*,n} = \arg\max_\alpha F\big(x^{n-1} + \alpha \nabla F(x^{n-1})\big).$$

We would then update x^n using $\alpha_{n-1} = \alpha^{*,n}$.

There are, of course, problems where x has to stay in a feasible region $\mathcal{X}$, which might be described by a system of linear equations such as

$$\mathcal{X} = \{x | Ax = b, x \geq 0\}.$$

When we have constraints, we can run our update in (5.10) through a projection step that we write using

$$x^n \leftarrow \Pi_{\mathcal{X}}[x^{n-1} + \alpha_{n-1} \nabla_x F(x^{n-1})].$$

The definition of the projection operator $\Pi_{\mathcal{X}}[\cdot]$ is given by

$$\Pi_{\mathcal{X}}[x] = \arg\min_{x' \in \mathcal{X}} \|x - x'\|_2, \tag{5.11}$$

where $\|x - x'\|_2$ is the "L_2 norm" defined by

$$\|x - x'\|_2 = \sum_i (x_i - x_i')^2.$$

The projection operator $\Pi_{\mathcal{X}}[\cdot]$ can often be solved easily by taking advantage of the structure of a problem. For example, we may have box constraints of the form $0 \leq x_i \leq u_i$. In this case, any element x_i falling outside of this range is just mapped back to the nearest boundary (0 or u_i).

We assume in our work that we cannot compute the expectation exactly. Instead, we resort to an algorithmic strategy known as stochastic gradients, but also known as stochastic approximation procedures. We first present this idea in the classical asymptotic setting, and then revisit it for the more practical finite budget setting.

5.3.1 A stochastic gradient algorithm - asymptotic analysis

For our stochastic problem, we assume that we either cannot compute $F(x)$, or we cannot compute the gradient exactly. However, there are many problems where, if we fix $W = W(\omega)$, we can find the derivative of $F(x, W(\omega))$ with respect to x. Then, instead of using the deterministic updating formula in (5.10), we would instead use

$$x^n = x^{n-1} + \alpha_{n-1} \nabla_x F(x^{n-1}, W^n). \tag{5.12}$$

Here, $\nabla_x F(x^{n-1}, W^n)$ is called a *stochastic gradient* because it depends on a sample realization of W^n. It is important to note our indexing. A variable such as x^{n-1} or α_{n-1} that is indexed by $n-1$ is assumed to be a function of the observations $W^1, W^2, \ldots, W^{n-1}$, but not W^n. Thus, our stochastic gradient $\nabla_x F(x^{n-1}, W^n)$ depends on our previous solution x^{n-1} and our most recent observation W^n. To illustrate, consider the simple newsvendor problem which objective

$$F(x, W) = p \min\{x, W\} - cx.$$

In this problem, we order a quantity $x = x^{n-1}$ (determined at the end of day $n-1$, and then observe a random demand W^n that was observed on day n. We earn a revenue given by $p \min\{x^{n-1}, W^n\}$ (we cannot sell more than we bought, or more than the demand), but we had to pay for our order, producing a negative cost cx. Let $\nabla F(x^{n-1}, W^n)$ be the sample gradient, taken when $W = W^n$. In our example, this is given by

$$\frac{\partial F(x^{n-1}, W^n)}{\partial x} = \begin{cases} p - c & \text{If } x^{n-1} < W^n, \\ -c & \text{If } x^{n-1} > W^n. \end{cases} \tag{5.13}$$

The quantity x^{n-1} is the estimate of x computed from the previous iteration (using the sample realization ω^{n-1}, while W^n is the sample realization in iteration n (the indexing tells us that x^{n-1} was computed without knowing W^n). When the function is deterministic, we would choose the stepsize by solving the one-dimensional optimization problem

$$\max_\alpha F\big(x^{n-1} + \alpha_{n-1} \nabla F(x^{n-1}, W^n)\big).$$

Now we face the problem of finding the stepsize α_{n-1}. Unlike our deterministic algorithm, we cannot solving a one-dimensional search to find the best stepsize. Part of the problem is that a stochastic gradient can even point away from the optimal solution such that any positive stepsize actually makes the solution worse. For example, the right order quantity might be 15. However, even if we order $x = 20$, it is possible that the demand is greater 20 on a particular day. Although our optimal solution is less than our current estimate, the algorithm tells us to increase x.

Remark: Many authors will write equation (5.10) in the form

$$x^{n+1} = x^n + \alpha_n \nabla F(x^n, W^n). \tag{5.14}$$

With this style, we would say that x^{n+1} is the estimate of x to be used in iteration $n+1$ (although it was computed with the information from iteration n). We use the form in (5.10) because we will later allow the stepsizes to depend on the data, and the indexing tells us the information content of the stepsize. For theoretical reasons, it is important that the stepsize be computed using information up through $n-1$, hence our use of α_{n-1}. We index x^n on the left-hand side of (5.10) using n because the right-hand side has information from iteration n. It is often the case that time t is also our iteration counter, and so it helps to be consistent with our time indexing notation.

There are many applications where the units of the gradient, and the units of the decision variable, are different. This happens with our newsvendor example, where the gradient is in units of dollars, while the decision variable x is in units of newspapers. This is a significant problem that causes headaches in practice.

A problem where we avoid this issue arises if we are trying to learn the mean of a random variable W. We can formulate this task as a stochastic optimization problem using

$$\min_x \mathbb{E}\frac{1}{2}(x - W)^2. \tag{5.15}$$

Here, our function $F(x, W) = \frac{1}{2}(x - W)^2$, and it is not hard to see that the value of x that minimizes this function is $x = \mathbb{E}W$. Now assume that we want to produce a sequence of estimates of $\mathbb{E}W$ by solving this problem using a sequential (online) stochastic gradient algorithm, which looks like

$$\begin{aligned} x^{n+1} &= x^n - \alpha_n \nabla F_x(x^n, W^{n+1}), && (5.16)\\ &= x^n - \alpha_n(x^n - W^{n+1}), \\ &= (1-\alpha_n)x^n + \alpha_n W^{n+1}. && (5.17) \end{aligned}$$

Equation 5.16 illustrates α_n as the stepsize in a stochastic gradient algorithm, while equation (5.17) illustrates α_n as what is widely known as a "learning rate."

There are problems where we may start with a prior estimate of $\mathbb{E}W$ which we can express as x^0. In this case, we would want to use an initial stepsize $\alpha^0 < 1$. However, we often start with no information, in which case an initial stepsize $\alpha^0 = 1$ gives us

$$\begin{aligned} x^1 &= (1-\alpha_0)x^0 + \alpha_0 W^1 \\ &= W^1, \end{aligned}$$

which means we do not need the initial estimate for x^0. Smaller initial stepsizes would only make sense if we had access to a reliable initial guess, and in this case, the stepsize should reflect the confidence in our original estimate (for example, we might be warm starting an algorithm from a previous iteration).

We can evaluate our performance using a mean squared statistical measure. If we have an initial estimate x^0, we would use

$$MSE = \frac{1}{n}\sum_{m=1}^{n}(x^{m-1} - W^m)^2. \tag{5.18}$$

However, it is often the case that the sequence of random variables W^n is nonstationary, which means they are coming from a distribution that is changing over time. In this case, estimating the mean squared error is similar to our problem of estimating the mean of the random variable W, in which case we should use a standard stochastic gradient (smoothing) expression of the form

$$MSE^n = (1 - \beta_{n-1})MSE^{n-1} + \beta_{n-1}(x^{n-1} - W^n)^2,$$

where β_{n-1} is another stepsize sequence (which could be the same as α_{n-1}).

5.3.2 A note on notation

Throughout this volume, we index variables (whether we are indexing by iterations or time) to clearly identify the *information content* of each variable. Thus, x^n is the decision made after W^n becomes known. When we compute our stochastic gradient $\nabla_x F(x^{n-1}, W^n)$, we use x^{n-1} which was determined after observing W^{n-1}. If the iteration counter refers to an experiment, then it means that x^{n-1} is determined after we finish the $n-1st$ experiment. If we are solving a newsvendor problem where n indexes days, then it is like determining the amount of newspapers to order for day n after observing the sales for day $n-1$. If we are performing a laboratory experiment, we use the information up through the first $n-1$ experiments to choose x^{n-1}, which specifies the choice of the nth experiment. This indexing makes sense when you realize that the index $n-1$ reflects the information content, not when it is being implemented.

In chapter 6, we are going to present a number of formulas to determine stepsizes. Some of these are deterministic, such as $\alpha_n = 1/n$, and some are stochastic, adapting to information as it arrives. Our stochastic gradient formula in equation (5.12) communicates the property that the stepsize α_{n-1} that is multiplied times the gradient $\nabla_x F(x^{n-1}, W^n)$ is allowed to see W^{n-1} and x^{n-1}, but not W^n.

We return to this issue in chapter 9, but we urge readers to adopt this notational system.

5.3.3 A finite horizon formulation

The formulation in (5.8) assumes that we can test different values of x, after which we can observe $F(x, W)$, where the only goal is to find a single, deterministic x that solves (5.8). We only care about the final solution, not the quality of different values of x that we test while finding the optimal solution.

We have seen that an asymptotic analysis of (5.8) using a stochastic gradient algorithm such as (5.12) approaches the problem just as we would any deterministic optimization problem. In practice, of course, we have to limit our evaluation of different algorithms based on how well they do within some specified budget. In this section, we are going to see what happens when we formally approach the problem within a fixed budget, known in some communities as *finite time analysis*.

We begin by assuming that $F(x, W)$ is a reward that we incur each time we test x. For example, we might have a newsvendor problem of the form

$$F(x, W) = p \min\{x, W\} - cx.$$

Here, W would be interpreted as a demand and x is the supply, where p is the price at which newspapers are sold, and c is the purchase cost. We can use our newsvendor problem for computing the stochastic gradients we need in our stochastic gradient algorithm (5.12).

In practice we have to tune the stepsize formula. While there are many rules we could use, we will illustrate the key idea using a simple stepsize rule known as Kesten's rule given by

$$\alpha_n(\theta) = \frac{\theta}{\theta + N^n},$$

where θ is a tunable parameter and N^n counts the number of times that the objective function $F(x^n, W^{n+1})$ has declined. If we are steadily improving, we do not want to reduce the stepsize. If the objective function gets worse, then it is an indication that we are in the vicinity of the optimum, and we want to start using smaller stepsizes.

Now, our stochastic gradient algorithm (5.12) becomes a policy $X^\pi(S^n)$ with state $S^n = (x^n, N^n)$, and where π captures the structure of the rule (e.g. the stepsize rule in (5.12)) and any tunable parameters (that is, θ). Let $x^{\pi,N}$ be the solution x^n for $n = N$, where we include π to indicate that our solution $x^{\pi,N}$ after N function evaluations depends on the policy π that we followed to get there. The problem of finding the best stepsize rule can be stated as

$$\max_\pi \mathbb{E} F(x^{\pi,N}, W), \tag{5.19}$$

which states that we are looking for the best terminal value within a budget of N function evaluations. Of course, we do not have to limit our search over policies to comparing stepsize rules. There are different ways of computing the gradient which we review below.

There is a substantial literature on stochastic optimization algorithms which prove asymptotic convergence, and then examine the rate of convergence empirically. Experimental testing of different algorithms is forced to work with fixed budgets, which means that researchers are looking for the best solution within some budget. We would argue that (5.19) is stating the aspirational goal of finding the *optimal algorithm* for maximizing $\mathbb{E}F(x, W)$ in N iterations.

5.3.4 Stepsizes

We have rather casually introduced the idea of the stepsize α_n which pervades all gradient-based algorithms. In the previous section, we have presented the idea that the stepsize is actually a decision we have to make, and we need a rule that we might call a stepsize policy to make this decision.

Stepsizes are the price we pay for the simplicity of gradient-based methods. The issues are so rich we have dedicated an entire chapter to the topic (chapter 6). At this time we only offer the following word of caution: stepsizes all require tuning, which we present as a form of policy search. In practice, searching for the best stepsize policy is an annoying but unavoidable part of the algorithmic design process. For this reason, chapter 6 presents some formulas that require little or no tuning (depending on the setting).

5.4 STYLES OF GRADIENTS

There are a few variants of the basic stochastic gradient method. Below we introduce the idea of gradient smoothing and describe a method for approximating a second-order algorithm.

5.4.1 Gradient smoothing

In practice, stochastic gradients can be *highly* stochastic, which is the reason why we have to use stepsizes. However, it is possible to mitigate some of the variability by smoothing the gradient itself. If $\nabla F(x^n, W^{n+1})$ is our stochastic gradient, computed after the $n+1st$ experiment, we could then smooth this using

$$g^{n+1} = (1-\eta)g^n + \eta \nabla F(x^n, W^{n+1}),$$

where η is a smoothing factor where $0 < \eta \leq 1$. We could replace this with a declining sequence η_n, although common practice is to keep this process as simple as possible. Regardless of the strategy, gradient smoothing has the effect of introducing at least one more tunable parameter. The open empirical question is whether gradient smoothing adds anything beyond the smoothing produced by the stepsize policy used for updating x^n.

5.4.2 Second-order methods

Second order methods for deterministic optimization have proven to be particularly attractive. For smooth, differentiable functions, the basic update step looks like

$$x^{n+1} = x^n + (H^n)^{-1} \nabla_x f(x^n), \tag{5.20}$$

where H^n is the Hessian, which is the matrix of second derivatives. That is,

$$H^n_{xx'} = \left. \frac{\partial^2 f(x)}{\partial x \partial x'} \right|_{x=x^n}.$$

The attraction of the update in equation (5.20) is that there is no stepsize. The reason (and this requires that $f(x)$ be smooth with continuous first derivatives) is that the inverse Hessian solves the problem of scaling. In fact, if $f(x)$ is quadratic, then equation (5.20) takes us to the optimal solution in one step!

Since functions are not always as nice as we would like, it is sometimes useful to introduce a constant "stepsize" α, giving us

$$x^{n+1} = x^n + \alpha (H^n)^{-1} \nabla_x f(x^n),$$

where $0 < \alpha \leq 1$. Note that this smoothing factor does not have to solve any scaling problems (again, this is solved by the Hessian).

If we have access to second derivatives (which is not always the case), then our only challenge is inverting the Hessian. This is not a problem with a few dozen or even a few hundred variables, but there are problems with thousands to tens of thousands of variables. For large problems, we can strike a compromise and just use the diagonal of the Hessian. This is both much easier to compute, as well as being easy to invert. Of course, we lose some of the fast convergence (and scaling).

There are many problems (including all stochastic optimization problems) where we do not have access to Hessians. One strategy to overcome this is to construct an approximation of the Hessian using what are known as rank-one updates. Let $\bar{H}^n$ be our approximate Hessian which is computed using

$$\bar{H}^{n+1} = \bar{H}^n + \nabla f(x^n)(\nabla f(x^n))^T. \tag{5.21}$$

Recall that $\nabla f(x^n)$ is a column vector, so $\nabla f(x^n)(\nabla f(x^n))^T$ is a matrix with the dimensionality of x. Since it is made up of an outer product of two vectors, this matrix has rank 1.

This methodology could be applied to a stochastic problem. As of this writing, we are not aware of any empirical study showing that these methods work, although there has been recent interest in second-order methods for online machine learning.

5.4.3 Finite differences

There are instances where we do not have direct access to a derivative, but where it makes sense to estimate the derivative using finite differences. In this setting, we can approximate gradients using finite differences. Assume that x is a K-dimensional vector, and let e_k be a K-dimensional column vector of zeroes with a 1 in the kth position. Now assume that we can run two simulations for each dimension, $F(x^n + \delta x^n e_k, W_k^{n+1,+})$ and $F(x^n - \delta x^n e_k, W_k^{n+1,-})$ where $\delta x^n e_k$ is the change in x^n, multiplied by e_k so that we are only changing the kth dimension. We use $W_k^{n+1,+}$ and $W_k^{n+1,}$ to represent the sequences of random variables that are generated when we run each simulation, which would be run in the $n+1st$ iteration. Think of $F(x^n + \delta x^n e_k, W_k^{n+1,+})$ and $F(x^n - \delta x^n e_k, W_k^{n+1,-})$ as calls to a black-box simulator where we start with a set of parameters x^n, and then perturb it to $x^n + \delta x^n e_k$ and $x^n - \delta x^n e_k$ and run two separate, independent simulations. We then have to do this for each dimension k, allowing us to compute

$$g_k^n(x^n, W^{n+1,+}, W^{n+1,-}) \quad = \quad \frac{F(x^n + \delta x^n e_k, W_k^{n+1,+}) - F(x^n - \delta x^n e_k, W_k^{n+1,-})}{2\delta x^n}, \tag{5.22}$$

where we divide the difference by the width of the change which is $2\delta x^n$ to get the slope.

The calculation of the derivative (for one dimension) is illustrated in figure 5.1. We see from figure 5.1 that shrinking δx can introduce a lot of noise in the estimate of the gradient. At the same time, as we increase δx, we introduce bias, which we see in the difference between the dashed line showing $\mathbb{E}g^n(x^n, W^{n+1,+}, W^{n+1,-})$, and the dotted line that depicts $\partial \mathbb{E}F(x^n, W^{n+1})/\partial x^n$. If we want an algorithm that converges asymptotically in the limit, we need δx^n decreasing, but in practice it is often set to a constant δx, which is then handled as a tunable parameter.

Finite differences can be expensive. Running a function evaluation can require seconds to minutes, but there are computer models that can take hours (or more) to run. Equation (5.22) requires $2K$ function evaluations, which can be especially problematic when $F(x, W)$ is an expensive simulation, as well as when the number of dimensions K is large.

There are several strategies we can pursue to reduce this computational burden:

- Instead of perturbing x^n up and down, just do it in one direction (typically up, unless a dimension of x^n is up against a constraint). This means we have to do $K+1$ function evaluations: a base estimate, and then one for each dimension.

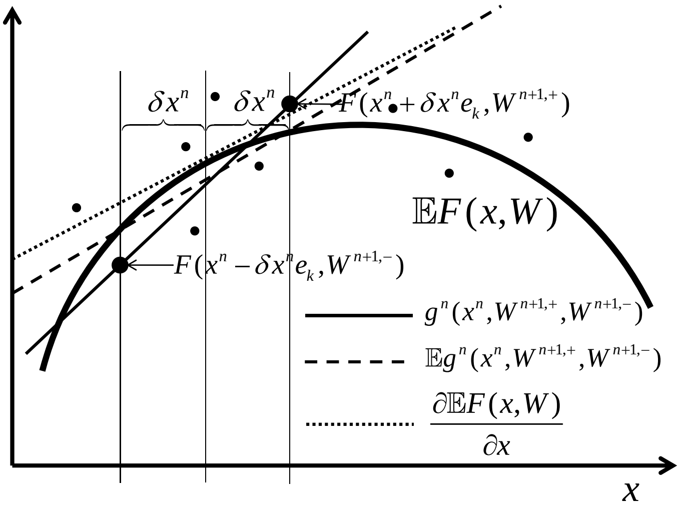

δx^n
δx^n
$F(x^n + \delta x^n e_k, W^{n+1,+})$
$\mathbb{E}F(x,W)$
$F(x^n - \delta x^n e_k, W^{n+1,-})$
$g^n(x^n, W^{n+1,+}, W^{n+1,-})$
$\mathbb{E}g^n(x^n, W^{n+1,+}, W^{n+1,-})$
$\frac{\partial \mathbb{E}F(x,W)}{\partial x}$
x

Figure 5.1 Different estimates of the gradient of $F(x, W)$ with a) the stochastic gradient $g^n(x^n, W^{n+1,+}, W^{n+1,-})$ (solid line), the expected finite difference $\mathbb{E}g^n(x^n, W^{n+1,+}, W^{n+1,-})$ (dashed line), and the exact slope at x^n, $\partial \mathbb{E}F(x^n, W^{n+1})/\partial x^n$.

- Just estimate the gradient for one dimension, where you would typically choose the dimension at random.
- Randomly perturb all the dimensions all at once. This is known as the *simultaneous perturbation stochastic approximation* procedure (or SPSA).

SPSA computes gradients in the following way. Let $Z_k, k = 1, \dots, K$ be a vector of zero-mean random variables, and let Z^n be a sample of this vector at iteration n. We approximate the gradient by perturbing x^n by the vector Z using $x^n + Z^n$ and $x^n - Z^n$. Now let $W^{n+1,+}$ and $W^{n+1,-}$ represent two different samples of the random variables driving the simulation (these can be generated in advance or on the fly). We then run our simulation twice: once to find $F(x^n + Z^n, W^{n+1,+})$, and once to find $F(x^n - Z^n, W^{n+1,-})$.

$$g^n(x^n, W^{n+1,+}, W^{n+1,-}) = \frac{F(x^n + Z^n, W^{n+1,+}) - F(x^n - Z^n, W^{n-1,-})}{2c^n} \begin{pmatrix} Z_1^n \\ Z_2^n \\ \vdots \\ Z_K^n \end{pmatrix} \tag{5.23}$$

SPSA is a powerful strategy for performing finite-different estimates for high-dimensional parameter vectors θ. SPSA has been found to provide the same level of statistical accuracy as traditional finite difference methods, but requiring only 2 function evaluations instead of $2K$ evaluations. This can be a dramatic savings for settings where the simulations are expensive.

5.5 TRANSIENT PROBLEMS

There are many applications where we are trying to solve our basic stochastic optimization problem in an online setting, where the random variable W comes from field observations.

In these settings, it is not unusual to find that the underlying distribution describing W is changing over time. For example, the demands in our newsvendor application may be changing as the purchasing patterns of the market change.

We tend to design algorithms so they exhibit asymptotic convergence. For example, we would insist that the stepsize α_n decline to zero as the algorithm progresses. In a transient setting, this is problematic because it means we are putting decreasing emphasis on the latest information, which is more important than older information. Over time, as α_n approaches zero, the algorithm will stop responding to new information. If we use a stepsize such as $\alpha_n = 1/n$, it is possibly to show that the algorithm will eventually adapt to new information, but the rate of adaptation is so slow that the results are not useful.

Practitioners avoid this problem by either choosing a constant stepsize, or one that starts large but converges to a constant greater than zero. If we do this, the algorithm will start bouncing around the optimum. While this behavior may seem undesirable, in practice this is preferable, partly because the optimum of stochastic optimization problems tend to be smooth, but mostly because it means the algorithm is still adapting to new information, which makes it responsive to a changing signal.

5.6 RECURSIVE BENDERS DECOMPOSITION FOR CONVEX PROBLEMS

We first saw Benders decomposition for a two-stage stochastic optimization model for a sampled problem in section 4.3.2. Here, we present an asymptotic version of Benders that is in the theme of the other iterative algorithms presented in this chapter. This version was first introduced as *stochastic decomposition*. We begin by introducing the basic algorithm, followed by a variant known as regularization that has been found to stabilize performance.

5.6.1 The basic algorithm

We begin by presenting the two-stage stochastic programming model we first presented in section 4.3.2:

$$\max_{x_0} \big(c_0 x_0 + \mathbb{E}Q_1(x_0, W)\big), \tag{5.24}$$

subject to

$$\begin{aligned} A_0 x_0 &= b, & (5.25)\\ x_0 &\geq 0. & (5.26)\end{aligned}$$

We are going to again solve the original problem (5.24) using a series of Benders cuts, but this time we are going to construct them somewhat different. The approximated problem still looks like

$$x^n = \arg\max_{x_0, z}(c_0 x_0 + z), \tag{5.27}$$

subject to (5.25)-(5.26) and

$$z \leq \alpha^n_m + \beta^n_m x_0, \;\; m = 1, \ldots, n-1. \tag{5.28}$$

Of course, for iteration $n = 1$ we do not have any cuts.

The second stage problem which is solved for a given value $W(\omega)$ which specifies the costs and the demand D_1. In our iterative algorithm, we solve the problem for ω^n, using the solution x_0^n from the first stage:

$$Q_1(x_0^n, \omega^n) = \max_{x_1(\omega^n)} c_1(\omega^n)x_1(\omega^n), \tag{5.29}$$

subject to:

$$\begin{aligned} A_1 x_1(\omega^n) &\leq B_1 x_0^n, & (5.30)\\ B_1 x_1(\omega^n) &\leq D_1(\omega^n), & (5.31)\\ x_1(\omega^n) &\geq 0.. & (5.32)\end{aligned}$$

As before, we let $\hat{\beta}^n$ be the dual variable for the resource constraint (5.30) when we solve the problem using sample ω^n. Then let

$$\begin{aligned} \alpha_n^n &= \frac{1}{n}\sum_{m=1}^{n} Q_1(x_0^m, \omega^m),\\ \beta_n^n &= \frac{1}{n}\sum_{m=1}^{n} \hat{\beta}^m. \end{aligned}$$

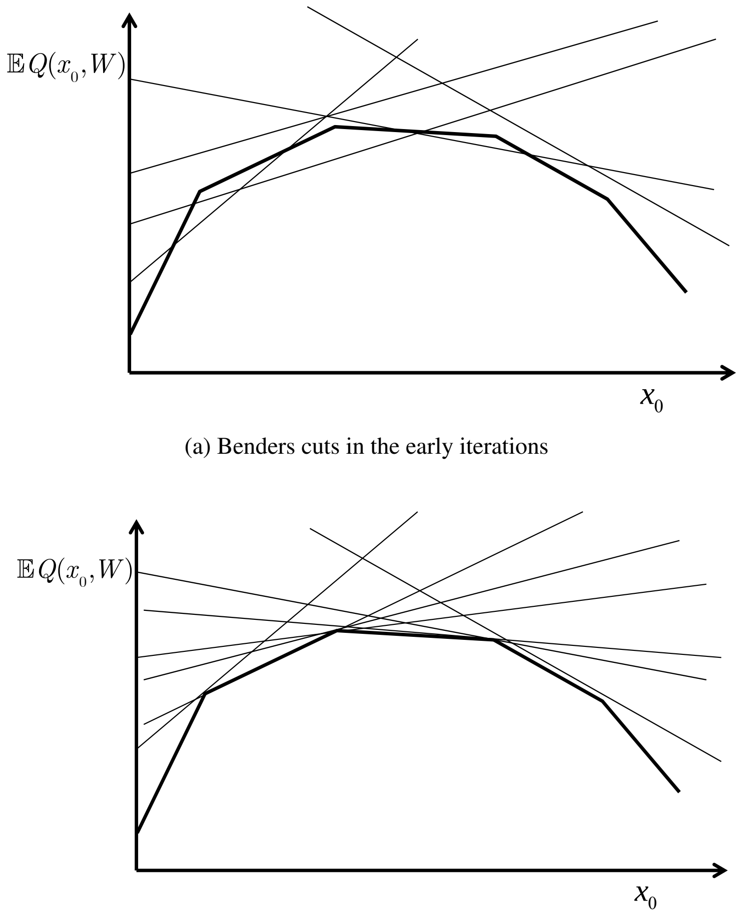
Thus, we compute α_n^n by averaging all the prior objective functions for the second stage, and then we compute β_n^n by averaging all the prior dual variables. We finally update all prior α_m^n and β_m^n for $m < n$ using

$$\begin{aligned} \alpha_m^n &= \frac{n-1}{n}\alpha_m^{n-1}, \;\; m = 1, \dots, n-1,\\ \beta_m^n &= \frac{n-1}{n}\beta_m^{n-1}, \;\; m = 1, \dots, n-1. \end{aligned}$$

Aside from the differences in how the Benders cuts are computed, the major difference between this implementation and our earlier sampled solution given in section 4.3.2 is that in this recursive formulation, the samples ω are drawn from the full sample space Ω rather than a sampled one. When we solve the sampled version of the problem, we solve it exactly in a finite number of iterations, but we only obtain an optimal solution to a sampled problem. Here, we have an algorithm that will asymptotically converge to the optimal solution of the original problem.

Figure 5.2 illustrates the cuts generated using stochastic decomposition. It is useful to compare the cuts generated using stochastic decomposition to those generated when we used a sampled version of the problem in section 4.3.2 as depicted in figure 4.1. When we were solving our sampled version, we could compute the expectation exactly, which is why the cuts were tight. Here, we are sampling from the full probability space, and as a result we get cuts that approximate the function, but nothing more. However, in the limit as $n \to \infty$, the cuts will converge to the true function in the vicinity of the optimum.

Which is better? Hard to say. While it is nice to have an algorithm that is asymptotically optimal, we can only run a finite number of iterations. The sampled problem will be more stable due to the averaging that takes place in every iteration, but we then have to solve a linear program for every ω in the sampled problem, a step that involves much more computational overhead than the recursive version.



(a) Benders cuts in the early iterations

(b) Benders cuts in the limit

Figure 5.2 Illustration of cuts generated using stochastic decomposition (a) in the early iterations and (b) in the limit.

5.6.2 Benders with regularization

Regularization is a tool that will come up repeatedly when estimating functions from data. The same is true with Benders decomposition. Regularization is handled through a minor modification of the approximate optimization problem (5.27) which becomes

$$x^n = \arg\max_x \left(c_0 x_0 + z + \rho^n (x - \bar{x}^{n-1})^2\right), \tag{5.33}$$

which is solved subject to (5.25)-(5.26) and the Benders cut constraints (5.28). The parameter ρ^n is a decreasing sequence that needs to be scaled to handle the difference in the units between the costs and the term $(x - \bar{x}^{n-1})^2)$. $\bar{x}^{n-1}$ is the regularization term which is updated each iteration; the idea with regularization is to keep x^n from straying too far from a previous solution, especially in the early iterations.

The use of the squared deviation $(x - \bar{x}^{n-1})^2$ is known as L_2 regularization, which might be written as $\|x - \bar{x}^{n-1}\|_2^2$. An alternative is L_1 regularization which minimizes the absolute value of the deviation, which would be written as $|x - \bar{x}^{n-1}|$.

There are different ways of setting the regularization term, but the simplest one just uses $\bar{x}^{n-1} = x^{n-1}$. Other ideas involve smoothing several previous iterations. The regularization coefficient is any declining sequence such as

$$\rho^k = r\rho^{k-1}$$

for some factor $r < 1$, starting with an initial ρ^0 that has to be chosen to handle the scaling.

Properly implemented, regularization offers not only theoretical guarantees, but has also been found to accelerate convergence and stabilize the performance of the algorithm.

5.7 EMPIRICAL ISSUES

Invariably, the process of actually implementing these algorithms raises issues that are often ignored when describing the algorithms. To help mitigate this transition, below are some of the challenges an experimentalist is likely to encounter.

Tunable parameters - Arguably one of the most frustrating aspects of any algorithms is the need to tune parameters. For gradient-based algorithms, this typically refers to the tunable parameters in the stepsize policy. These tunable parameters are a direct result of the use of first-order algorithms, which are easy to compute but which exploit very little information about the underlying function. Particularly frustrating is that this tuning really matters. A poorly tuned stepsize algorithm may decrease too quickly, creating apparent convergence. It is completely possible that a poorly tuned stepsize policy can result in a conclusion that an algorithm is not working. Stepsizes that are too large can introduce too much noise.

Scaling - In most (but not all) applications, the units of the gradient $\nabla F(x, W)$ are different than the units of x. A rough rule is that the initial stepsize should be chosen so that the initial change in x is on the order of 30 to 50 percent of the starting value.

Benchmarking - Whenever possible, it helps to run an algorithm on a simpler problem where the optimal solution can be found using other means, either analytically or numerically. For example, it might be possible to apply the stochastic gradient algorithm on a deterministic sequence that can be solved using deterministic algorithms.

Robustness - A desirable property of any algorithm is that it work reliably, on any problem instance (that is, within a problem class). For example, tuning parameters in the stepsize policy is annoying, but bearable if it only has to be done once.

5.8 WHY DOES IT WORK?**

Stochastic approximation methods have a rich history starting with the seminal paper Robbins & Monro (1951) and followed by Blum (1954*b*) and Dvoretzky (1956). The serious reader should see Kushner & Yin (1997) for a modern treatment of the subject. Wasan (1969) is also a useful reference for fundamental results on stochastic convergence theory. A separate line of investigation was undertaken by researchers in eastern European community focusing on constrained stochastic optimization problems (Gaivoronski (1988), Ermoliev (1988), Ruszczyński (1980), Ruszczyński (1987)). This work is critical to our fundamental understanding of Monte Carlo-based stochastic learning methods.

The theory behind these proofs is fairly deep and requires some mathematical maturity. For pedagogical reasons, we start in section 5.8.1 with some probabilistic preliminaries, after which section 5.8.2 presents one of the original proofs, which is relatively more accessible and which provides the basis for the universal requirements that stepsizes must satisfy for theoretical proofs. Section 5.8.3 provides a more modern proof based on the theory of martingales.

5.8.1 Some probabilistic preliminaries

The goal in this section is to prove that these algorithms work. But what does this mean? The solution $\bar{x}^n$ at iteration n is a random variable. Its value depends on the sequence of sample realizations of the random variables over iterations 1 to n. If $\omega = (W^1, W^2, \ldots, W^n, \ldots)$ represents the sample path that we are following, we can ask what is happening to the limit $\lim_{n\rightarrow\infty} \bar{x}^n(\omega)$. If the limit is x^*, does x^* depend on the sample path ω?

In the proofs below, we show that the algorithms converge *almost surely*. What this means is that

$$\lim_{n\rightarrow\infty} \bar{x}^n(\omega) = x^*$$

for all $\omega \in \Omega$ that can occur with positive measure. This is the same as saying that we reach x^* with probability 1. Here, x^* is a deterministic quantity that does not depend on the sample path. Because of the restriction $p(\omega) > 0$, we accept that in theory, there could exist a sample outcome that can never occur that would produce a path that converges to some other point. As a result, we say that the convergence is "almost sure," which is universally abbreviated as "*a.s.*" Almost sure convergence establishes the core theoretical property that the algorithm will eventually settle in on a single point. This is an important property for an algorithm, but it says nothing about the rate of convergence (an important issue in approximate dynamic programming).

Let $x \in \Re^n$. At each iteration n, we sample some random variables to compute the function (and its gradient). The sample realizations are denoted by W^n. We let $\omega = (W^1, W^2, \ldots,)$ be a realization of all the random variables over all iterations. Let Ω be the set of all possible realizations of ω, and let $\mathfrak{F}$ be the σ-algebra on Ω (that is to say, the set of all possible events that can be defined using Ω). We need the concept of the history up through iteration n. Let

$H^n =$ A random variable giving the history of all random variables up through iteration n.

A sample realization of H^n would be

$$\begin{aligned} h^n &= H^n(\omega) \\ &= (W^1, W^2, \ldots, W^n). \end{aligned}$$

We could then let W^n be the set of all outcomes of the history (that is, $h^n \in H^n$) and let $\mathcal{H}^n$ be the σ-algebra on W^n (which is the set of all events, including their complements and unions, defined using the outcomes in W^n). Although we could do this, this is not the convention followed in the probability community. Instead, we define a sequence of

σ-algebras $\mathfrak{F}^1, \mathfrak{F}^2, \ldots, \mathfrak{F}^n$ as the sequence of σ-algebras on Ω that can be generated as we have access to the information through the first $1, 2, \ldots, n$ iterations, respectively. What does this mean? Consider two outcomes $\omega \neq \omega'$ for which $H^n(\omega) = H^n(\omega')$. If this is the case, then any event in $\mathfrak{F}^n$ that includes ω must also include ω'. If we say that a function is $\mathfrak{F}^n$-measurable, then this means that it must be defined in terms of the events in $\mathfrak{F}^n$, which is in turn equivalent to saying that we cannot be using any information from iterations $n+1, n+2, \ldots$.

We would say, then, that we have a standard probability space $(\Omega, \mathfrak{F}, \mathcal{P})$ where $\omega \in \Omega$ represents an elementary outcome, $\mathfrak{F}$ is the σ-algebra on $\mathfrak{F}$ and $\mathcal{P}$ is a probability measure on Ω. Since our information is revealed iteration by iteration, we would also then say that we have an increasing set of σ-algebras $\mathfrak{F}^1 \subseteq \mathfrak{F}^2 \subseteq \ldots \subseteq \mathfrak{F}^n$ (which is the same as saying that $\mathcal{F}^n$ is a filtration).

5.8.2 An older proof

Enough with probabilistic preliminaries. We wish to solve the unconstrained problem

$$\max_x \mathbb{E}F(x, \omega) \tag{5.34}$$

with x^* being the optimal solution. Let $g(x, \omega)$ be a stochastic ascent vector that satisfies

$$g(x, \omega)^T \nabla F(x, \omega) \geq 0. \tag{5.35}$$

For many problems, the most natural ascent vector is the gradient itself

$$g(x, \omega) \quad = \quad \nabla F(x, \omega) \tag{5.36}$$

which clearly satisfies (5.35).

We assume that $F(x) = \mathbb{E}F(x, \omega)$ is continuously differentiable and concave, with bounded first and second derivatives so that for finite M

$$-M \leq g(x, \omega)^T \nabla^2 F(x) g(x, \omega) \leq M. \tag{5.37}$$

A stochastic gradient algorithm (sometimes called a stochastic approximation method) is given by

$$\bar{x}^n \quad = \quad \bar{x}^{n-1} + \alpha_{n-1} g(\bar{x}^{n-1}, \omega). \tag{5.38}$$

We first prove our result using the proof technique of Blum (1954*b*) that generalized the original stochastic approximation procedure proposed by Robbins & Monro (1951) to multidimensional problems. This approach does not depend on more advanced concepts such as martingales and, as a result, is accessible to a broader audience. This proof helps the reader understand the basis for the conditions $\sum_{n=0}^{\infty} \alpha_n = \infty$ and $\sum_{n=0}^{\infty} (\alpha_n)^2 < \infty$ that are required of all stochastic approximation algorithms.

We make the following (standard) assumptions on stepsizes

$$\alpha_n \quad > \quad 0 \text{ for all } n \geq 0, \tag{5.39}$$

$$\sum_{n=0}^{\infty} \alpha_n \quad = \quad \infty, \tag{5.40}$$

$$\sum_{n=0}^{\infty} (\alpha_n)^2 \quad < \quad \infty. \tag{5.41}$$

We want to show that under suitable assumptions, the sequence generated by (5.38) converges to an optimal solution. That is, we want to show that

$$\lim_{n\rightarrow\infty} x^n \quad = \quad x^* \ a.s. \tag{5.42}$$

We now use Taylor's theorem (remember Taylor's theorem from freshman calculus?), which says that for any continuously differentiable convex function $F(x)$, there exists a parameter $0 \leq \eta \leq 1$ that satisfies for a given x and x^0

$$F(x) \quad = \quad F(x^0) + \nabla F(x^0 + \eta(x - x^0))(x - x^0). \tag{5.43}$$

This is the first-order version of Taylor's theorem. The second-order version takes the form

$$F(x) \quad = \quad F(x^0) + \nabla F(x^0)(x - x^0) + \frac{1}{2}(x - x^0)^T \nabla^2 F(x^0 + \eta(x - x^0))(x - x^0) \tag{5.44}$$

for some $0 \leq \eta \leq 1$. We use the second-order version. In addition, since our problem is stochastic, we will replace $F(x)$ with $F(x, \omega)$ where ω tells us what sample path we are on, which in turn tells us the value of W.

To simplify our notation, we are going to replace x^0 with x^{n-1}, x with x^n, and finally we will use

$$g^n = g(x^{n-1}, \omega). \tag{5.45}$$

This means that, by definition of our algorithm,

$$\begin{aligned} x - x^0 &= x^n - x^{n-1} \\ &= (x^{n-1} + \alpha_{n-1} g^n) - x^{n-1} \\ &= \alpha_{n-1} g^n. \end{aligned}$$

From our stochastic gradient algorithm (5.38), we may write

$$\begin{aligned} F(x^n, \omega) &= F(x^{n-1} + \alpha_{n-1} g^n, \omega) \\ &= F(x^{n-1}, \omega) + \nabla F(x^{n-1}, \omega)(\alpha_{n-1} g^n) \\ &\quad + \frac{1}{2}(\alpha_{n-1} g^n)^T \nabla^2 F(x^{n-1} + \eta \alpha_{n-1} g^n, \omega)(\alpha_{n-1} g^n). \end{aligned} \tag{5.46}$$

It is now time to use a *standard mathematician's trick*. We sum both sides of (5.46) to get

$$\begin{aligned} \sum_{n=1}^{N} F(x^n, \omega) &= \sum_{n=1}^{N} F(x^{n-1}, \omega) + \sum_{n=1}^{N} \nabla F(x^{n-1}, \omega)(\alpha_{n-1} g^n) + \\ &\quad \frac{1}{2} \sum_{n=1}^{N} (\alpha_{n-1} g^n)^T \nabla^2 F\left(x^{n-1} + \eta \alpha_{n-1} g^n, \omega\right)(\alpha_{n-1} g^n). \end{aligned} \tag{5.47}$$

Note that the terms $F(x^n), n = 2, 3, \ldots, N$ appear on both sides of (5.47). We can cancel these. We then use our lower bound on the quadratic term (5.37) to write

$$F(x^N, \omega) \geq F(x^0, \omega) + \sum_{n=1}^{N} \nabla F(x^{n-1}, \omega)(\alpha_{n-1} g^n) + \frac{1}{2} \sum_{n=1}^{N} (\alpha_{n-1})^2 (-M). \tag{5.48}$$

We now want to take the limit of both sides of (5.48) as $N \to \infty$. In doing so, we want to show that everything must be bounded. We know that $F(x^N)$ is bounded (*almost surely*) because we assumed that the original function was bounded. We next use the assumption (5.41) that the infinite sum of the squares of the stepsizes is also bounded to conclude that the rightmost term in (5.48) is bounded. Finally, we use (5.35) to claim that all the terms in the remaining summation ($\sum_{n=1}^{N} \nabla F(x^{n-1})(\alpha_{n-1} g^n)$) are positive. That means that this term is also bounded (from both above and below).

What do we get with all this boundedness? Well, if

$$\sum_{n=1}^{\infty} \alpha_{n-1} \nabla F(x^n, \omega) g^n < \infty \text{ for all } \omega \tag{5.49}$$

and (from (5.40))

$$\sum_{n=1}^{\infty} \alpha_{n-1} = \infty. \tag{5.50}$$

We can conclude that

$$\sum_{n=1}^{\infty} \nabla F(x^{n-1}, \omega) g^n < \infty. \tag{5.51}$$

Since all the terms in (5.51) are positive, they must go to zero. (Remember, everything here is true *almost surely*; after a while, it gets a little boring to keep saying *almost surely* every time. It is a little like reading Chinese fortune cookies and adding the automatic phrase "under the sheets" at the end of every fortune.)

We are basically done except for some relatively difficult (albeit important if you are ever going to do your own proofs) technical points to really prove convergence. At this point, we would use technical conditions on the properties of our ascent vector g^n to argue that if $\nabla F(x^n, \omega) g^n \to 0$ then $\nabla F(x^n, \omega) \to 0$, (it is okay if g^n goes to zero as $F(x^n, \omega)$ goes to zero, but it cannot go to zero too quickly).

This proof was first proposed in the early 1950's by Robbins and Monro and became the basis of a large area of investigation under the heading of stochastic approximation methods. A separate community, growing out of the Soviet literature in the 1960's, addressed these problems under the name of stochastic gradient (or stochastic quasi-gradient) methods. More modern proofs are based on the use of martingale processes, which do not start with Taylor's formula and do not (always) need the continuity conditions that this approach needs.

Our presentation does, however, help to present several key ideas that are present in most proofs of this type. First, concepts of almost sure convergence are virtually standard. Second, it is common to set up equations such as (5.46) and then take a finite sum as in (5.47) using the alternating terms in the sum to cancel all but the first and last elements of the sequence of some function (in our case, $F(x^{n-1}, \omega)$). We then establish the boundedness of this expression as $N \to \infty$, which will require the assumption that $\sum_{n=1}^{\infty} (\alpha_{n-1})^2 < \infty$. Then, the assumption $\sum_{n=1}^{\infty} \alpha_{n-1} = \infty$ is used to show that if the remaining sum is bounded, then its terms must go to zero.

More modern proofs will use functions other than $F(x)$. Popular is the introduction of so-called Lyapunov functions, which are artificial functions that provide a measure of optimality. These functions are constructed for the purpose of the proof and play no role in the algorithm itself. For example, we might let $T^n = ||x^n - x^*||$ be the distance between our current solution x^n and the optimal solution. We will then try to show that T^n is suitably reduced to prove convergence. Since we do not know x^*, this is not a function we can actually measure, but it can be a useful device for proving that the algorithm actually converges.

It is important to realize that stochastic gradient algorithms of all forms do not guarantee an improvement in the objective function from one iteration to the next. First, a sample gradient g^n may represent an appropriate ascent vector for a sample of the function $F(x^n, \omega)$ but not for its expectation. In other words, randomness means that we may go in the wrong direction at any point in time. Second, our use of a nonoptimizing stepsize, such as $\alpha_{n-1} = 1/n$, means that even with a good ascent vector, we may step too far and actually end up with a lower value.

5.8.3 A more modern proof

Since the original work by Robbins and Monro, more powerful proof techniques have evolved. Below we illustrate a basic martingale proof of convergence. The concepts are somewhat more advanced, but the proof is more elegant and requires milder conditions. A significant generalization is that we no longer require that our function be differentiable (which our first proof required). For large classes of resource allocation problems, this is a significant improvement.

First, just what is a martingale? Let $\omega_1, \omega_2, \ldots, \omega_t$ be a set of exogenous random outcomes, and let $h_t = H_t(\omega) = (\omega_1, \omega_2, \ldots, \omega_t)$ represent the history of the process up to time t. We also let $\mathfrak{F}_t$ be the σ-algebra on Ω generated by H_t. Further, let U_t be a function that depends on h_t (we would say that U_t is a $\mathfrak{F}_t$-measurable function), and bounded ($\mathbb{E}|U_t| < \infty,\ \forall t \geq 0$). This means that if we know h_t, then we know U_t deterministically (needless to say, if we only know h_t, then U_{t+1} is still a random variable). We further assume that our function satisfies

$$\mathbb{E}[U_{t+1}|\mathfrak{F}_t] = U_t.$$

If this is the case, then we say that U_t is a *martingale*. Alternatively, if

$$\mathbb{E}[U_{t+1}|\mathfrak{F}_t] \leq U_t \tag{5.52}$$

then we say that U_t is a *supermartingale*. If U_t is a supermartingale, then it has the property that it drifts downward, usually to some limit point U^*. What is important is that it only drifts downward in expectation. That is, it could easily be the case that $U_{t+1} > U_t$ for specific outcomes. This captures the behavior of stochastic approximation algorithms. Properly designed, they provide solutions that improve on average, but where from one iteration to another the results can actually get worse.

Finally, assume that $U_t \geq 0$. If this is the case, we have a sequence U_t that drifts downward but which cannot go below zero. Not surprisingly, we obtain the following key result:

Theorem 5.8.1. *Let U_t be a positive supermartingale. Then, U_t converges to a finite random variable U^* almost surely.*

Note that "almost surely" (which is typically abbreviated "a.s.") means "for all (or every) ω." Mathematicians like to recognize every possibility, so they will add "every ω that might happen with some probability," which means that we are allowing for the possibility that U_t

might not converge for some sample realization ω that would never actually happen (that is, where $p(\omega) > 0$). This also means that it converges with probability one.

So what does this mean for us? We assume that we are still solving a problem of the form

$$\max_x \mathbb{E}F(x, \omega), \tag{5.53}$$

where we assume that $F(x, \omega)$ is continuous and concave (but we do not require differentiability). Let $\bar{x}^n$ be our estimate of x at iteration n (remember that $\bar{x}^n$ is a random variable). Instead of watching the evolution of a process of time, we are studying the behavior of an algorithm over iterations. Let $F^n = \mathbb{E}F(\bar{x}^n)$ be our objective function at iteration n and let F^* be the optimal value of the objective function. If we are maximizing, we know that $F^n \leq F^*$. If we let $U^n = F^* - F^n$, then we know that $U^n \geq 0$ (this assumes that we can find the true expectation, rather than some approximation of it). A stochastic algorithm will not guarantee that $F^n \geq F^{n-1}$, but if we have a good algorithm, then we may be able to show that U^n is a supermartingale, which at least tells us that in the limit, U^n will approach some limit $\bar{U}$. With additional work, we might be able to show that $\bar{U} = 0$, which means that we have found the optimal solution.

A common strategy is to define U^n as the distance between $\bar{x}^n$ and the optimal solution, which is to say

$$U^n = (\bar{x}^n - x^*)^2. \tag{5.54}$$

Of course, we do not know x^*, so we cannot actually compute U^n, but that is not really a problem for us (we are just trying to prove convergence). Note that we immediately get $U^n \geq 0$ (without an expectation). If we can show that U^n is a supermartingale, then we get the result that U^n converges to a random variable U^* (which means the algorithm converges). Showing that $U^* = 0$ means that our algorithm will (eventually) produce the optimal solution. We are going to study the convergence of our algorithm for maximizing $\mathbb{E}F(x, W)$ by studying the behavior of U^n.

We are solving this problem using a stochastic gradient algorithm

$$\bar{x}^n = \bar{x}^{n-1} + \alpha_{n-1} g^n, \tag{5.55}$$

where g^n is our stochastic gradient. If F is differentiable, we would write

$$g^n = \nabla_x F(\bar{x}^{n-1}, W^n).$$

But in general, F may be nondifferentiable, in which case we may have multiple gradients at a point $\bar{x}^{n-1}$ (for a single sample realization). In this case, we write

$$g^n \in \partial_x F(\bar{x}^{n-1}, W^n),$$

where $\partial_x F(\bar{x}^{n-1}, W^n)$ refers to the set of subgradients at $\bar{x}^{n-1}$. We assume our problem is unconstrained, so $\nabla_x F(\bar{x}^*, W^n) = 0$ if F is differentiable. If it is nondifferentiable, we would assume that $0 \in \partial_x F(\bar{x}^*, W^n)$.

Throughout our presentation, we assume that x (and hence g^n) is a scalar (exercise 6.12 provides an opportunity to redo this section using vector notation). In contrast with the previous section, we are now going to allow our stepsizes to be stochastic. For this reason, we need to slightly revise our original assumptions about stepsizes (equations (5.39) to

(5.41)) by assuming

$$\begin{aligned} \alpha_n &> 0 \ \textit{a.s.}, & (5.56)\\ \sum_{n=0}^{\infty} \alpha_n &= \infty \ \textit{a.s.}, & (5.57)\\ \mathbb{E}\left[\sum_{n=0}^{\infty} (\alpha_n)^2\right] &< \infty. & (5.58) \end{aligned}$$

The requirement that α_n be nonnegative "almost surely" (a.s.) recognizes that α_n is a random variable. We can write $\alpha_n(\omega)$ as a sample realization of the stepsize (that is, this is the stepsize at iteration n if we are following sample path ω). When we require that $\alpha_n \geq 0$ "almost surely" we mean that $\alpha_n(\omega) \geq 0$ for all ω where the probability (more precisely, probability measure) of ω, $p(\omega)$, is greater than zero (said differently, this means that the probability that $\mathbb{P}[\alpha_n \geq 0] = 1$). The same reasoning applies to the sum of the stepsizes given in equation (5.57). As the proof unfolds, we will see the reason for needing the conditions (and why they are stated as they are).

We next need to assume some properties of the stochastic gradient g^n. Specifically, we need to assume the following:

Assumption 1 - $\mathbb{E}[g^{n+1}(\bar{x}^n - x^*)|\mathfrak{F}^n] \geq 0$,

Assumption 2 - $|g^n| \leq B_g$,

Assumption 3 - For any x where $|x - x^*| > \delta,\ \delta > 0$, there exists $\epsilon > 0$ such that $\mathbb{E}[g^{n+1}|\mathfrak{F}^n] > \epsilon$.

Assumption 1 assumes that on average, the gradient g^n points toward the optimal solution x^*. This is easy to prove for deterministic, differentiable functions. While this may be harder to establish for stochastic problems or problems where $F(x)$ is nondifferentiable, we do not have to assume that $F(x)$ is differentiable. Nor do we assume that a particular gradient g^{n+1} moves toward the optimal solution (for a particular sample realization, it is entirely possible that we are going to move away from the optimal solution). Assumption 2 assumes that the gradient is bounded. Assumption 3 requires that the expected gradient cannot vanish at a nonoptimal value of x. This assumption will be satisfied for any concave function.

To show that U^n is a supermartingale, we start with

$$\begin{aligned} U^{n+1} - U^n &= (\bar{x}^{n+1} - x^*)^2 - (\bar{x}^n - x^*)^2 \\ &= \big((\bar{x}^n - \alpha_n g^{n+1}) - x^*\big)^2 - (\bar{x}^n - x^*)^2 \\ &= \big((\bar{x}^n - x^*)^2 - 2\alpha_n g^{n+1}(\bar{x}^n - x^*) + (\alpha_n g^{n+1})^2\big) - (\bar{x}^n - x^*)^2 \\ &= (\alpha_n g^{n+1})^2 - 2\alpha_n g^{n+1}(\bar{x}^n - x^*). & (5.59) \end{aligned}$$

Taking conditional expectations on both sides gives

$$\mathbb{E}[U^{n+1}|\mathfrak{F}^n] - \mathbb{E}[U^n|\mathfrak{F}^n] = \mathbb{E}[(\alpha_n g^{n+1})^2|\mathfrak{F}^n] - 2\mathbb{E}[\alpha_n g^{n+1}(\bar{x}^n - x^*)|\mathfrak{F}^n]. \quad (5.60)$$

We note that

$$\begin{aligned} \mathbb{E}[\alpha_n g^{n+1}(\bar{x}^n - x^*)|\mathfrak{F}^n] &= \alpha_n \mathbb{E}[g^{n+1}(\bar{x}^n - x^*)|\mathfrak{F}^n] & (5.61)\\ &\geq 0. & (5.62) \end{aligned}$$

Equation (5.61) is subtle but important, as it explains a critical piece of notation in this book. Keep in mind that we may be using a stochastic stepsize formula, which means that α_n is a random variable. We assume that α_n is $\mathfrak{F}^n$-measurable, which means that we are not allowed to use information from iteration $n+1$ to compute it. This is why we use α_{n-1} in updating equations such as equation (5.10) instead of α_n. When we condition on $\mathfrak{F}^n$ in equation (5.61), α_n is deterministic, allowing us to take it outside the expectation. This allows us to write the conditional expectation of the product of α_n and g^{n+1} as the product of the expectations. Equation (5.62) comes from Assumption 1 and the nonnegativity of the stepsizes.

Recognizing that $\mathbb{E}[U^n|\mathfrak{F}^n] = U^n$ (given $\mathfrak{F}^n$), we may rewrite (5.60) as

$$\begin{aligned}\mathbb{E}[U^{n+1}|\mathfrak{F}^n] &= U^n + \mathbb{E}[(\alpha_n g^{n+1})^2|\mathfrak{F}^n] - 2\mathbb{E}[\alpha_n g^{n+1}(\bar{x}^n - x^*)|\mathfrak{F}^n] \\ &\leq U^n + \mathbb{E}[(\alpha_n g^{n+1})^2|\mathfrak{F}^n]. \qquad (5.63)\end{aligned}$$

Because of the positive term on the right-hand side of (5.63), we cannot directly get the result that U^n is a supermartingale. But hope is not lost. We appeal to a neat little trick that works as follows. Let

$$W^n = \mathbb{E}[U^n + \sum_{m=n}^{\infty}(\alpha_m g^{m+1})^2|\mathfrak{F}^n]. \qquad (5.64)$$

We are going to show that W^n is a supermartingale. From its definition, we obtain

$$\begin{aligned}W^n &= \mathbb{E}[W^{n+1} + U^n - U^{n+1} + (\alpha_n g^{n+1})^2|\mathfrak{F}^n], \\ &= \mathbb{E}\left[W^{n+1}|\mathfrak{F}^n\right] + U^n - \mathbb{E}\left[U^{n+1}|\mathfrak{F}^n\right] + \mathbb{E}\left[(\alpha_n g^{n+1})^2|\mathfrak{F}^n\right]\end{aligned}$$

which is the same as

$$\mathbb{E}[W^{n+1}|\mathfrak{F}^n] = W^n - \underbrace{\left(U^n + \mathbb{E}\left[(\alpha_n g^{n+1})^2|\mathfrak{F}^n\right] - \mathbb{E}[U^{n+1}|\mathfrak{F}^n]\right)}_{I}.$$

We see from equation (5.63) that $I \geq 0$. Removing this term gives us the inequality

$$\mathbb{E}[W^{n+1}|\mathfrak{F}^n] \leq W^n. \qquad (5.65)$$

This means that W^n is a supermartingale. It turns out that this is all we really need because $\lim_{n\to\infty} W^n = \lim_{n\to\infty} U^n$. This means that

$$\lim_{n\to\infty} U^n \to U^* \qquad a.s. \qquad (5.66)$$

Now that we have the basic convergence of our algorithm, we have to ask: but what is it converging to? For this result, we return to equation (5.59) and sum it over the values $n = 0$ up to some number N, giving us

$$\sum_{n=0}^{N}(U^{n+1} - U^n) = \sum_{n=0}^{N}(\alpha_n g^{n+1})^2 - 2\sum_{n=0}^{N}\alpha_n g^{n+1}(\bar{x}^n - x^*). \qquad (5.67)$$

The left-hand side of (5.67) is an alternating sum (sometimes referred to as a telescoping sum), which means that every element cancels out except the first and the last, giving us

$$U^{N+1} - U^0 = \sum_{n=0}^{N}(\alpha_n g^{n+1})^2 - 2\sum_{n=0}^{N}\alpha_n g^{n+1}(\bar{x}^n - x^*).$$

Taking expectations of both sides gives

$$\mathbb{E}[U^{N+1} - U^0] = \mathbb{E}\left[\sum_{n=0}^{N}(\alpha_n g^{n+1})^2\right] - 2\mathbb{E}\left[\sum_{n=0}^{N}\alpha_n g^{n+1}(\bar{x}^n - x^*)\right]. \quad (5.68)$$

We want to take the limit of both sides as N goes to infinity. To do this, we have to appeal to the *Dominated Convergence Theorem* (DCT), which tells us that

$$\lim_{N\to\infty}\int_x f^n(x)dx = \int_x \left(\lim_{N\to\infty} f^n(x)\right) dx$$

if $|f^n(x)| \leq g(x)$ for some function $g(x)$ where

$$\int_x g(x)dx < \infty.$$

For our application, the integral represents the expectation (we would use a summation instead of the integral if x were discrete), which means that the DCT gives us the conditions needed to exchange the limit and the expectation. Above, we showed that $\mathbb{E}[U^{n+1}|\mathfrak{F}^n]$ is bounded (from (5.63) and the boundedness of U^0 and the gradient). This means that the right-hand side of (5.68) is also bounded for all n. The DCT then allows us to take the limit as N goes to infinity inside the expectations, giving us

$$U^* - U^0 \;=\; \mathbb{E}\left[\sum_{n=0}^{\infty}(\alpha_n g^{n+1})^2\right] - 2\mathbb{E}\left[\sum_{n=0}^{\infty}\alpha_n g^{n+1}(\bar{x}^n - x^*)\right].$$

We can rewrite the first term on the right-hand side as

$$\begin{aligned}\mathbb{E}\left[\sum_{n=0}^{\infty}(\alpha_n g^{n+1})^2\right] &\leq \mathbb{E}\left[\sum_{n=0}^{\infty}(\alpha_n)^2(B)^2\right] && (5.69)\\ &= B^2\,\mathbb{E}\left[\sum_{n=0}^{\infty}(\alpha_n)^2\right] && (5.70)\\ &< \infty. && (5.71)\end{aligned}$$

Equation (5.69) comes from Assumption 2 which requires that $|g^n|$ be bounded by B, which immediately gives us Equation (5.70). The requirement that $\mathbb{E}\sum_{n=0}^{\infty}(\alpha_n)^2 < \infty$ (equation (5.41)) gives us (5.71), which means that the first summation on the right-hand side of (5.68) is bounded. Since the left-hand side of (5.68) is bounded, we can conclude that the second term on the right-hand side of (5.68) is also bounded.

Now let

$$\begin{aligned}\beta^n &= \mathbb{E}\left[g^{n+1}(\bar{x}^n - x^*)\right]\\ &= \mathbb{E}\left[\mathbb{E}\left[g^{n+1}(\bar{x}^n - x^*)|\mathfrak{F}^n\right]\right]\\ &\geq 0,\end{aligned}$$

since $\mathbb{E}[g^{n+1}(\bar{x}^n - x^*)|\mathfrak{F}^n] \geq 0$ from Assumption 1. This means that

$$\sum_{n=0}^{\infty}\alpha_n\beta^n \;<\; \infty \quad \text{a.s.} \qquad (5.72)$$

But, we have required that $\sum_{n=0}^{\infty} \alpha_n = \infty$ a.s. (equation (5.57)). Since $\alpha_n \geq 0$ and $\beta^n \geq 0$ (a.s.), we conclude that

$$\lim_{n \to \infty} \beta^n \quad \to \quad 0 \text{ a.s.} \tag{5.73}$$

If $\beta^n \to 0$, then $\mathbb{E}[g^{n+1}(\bar{x}^n - x^*)] \to 0$, which allows us to conclude that $\mathbb{E}[g^{n+1}(\bar{x}^n - x^*)|\mathfrak{F}^n] \to 0$ (the expectation of a nonnegative random variable cannot be zero unless the random variable is always zero). But what does this tell us about the behavior of $\bar{x}^n$? Knowing that $\beta^n \to 0$ does not necessarily imply that $g^{n+1} \to 0$ or $\bar{x}^n \to x^*$. There are three scenarios:

1) $\bar{x}^n \to x^*$ for all n, and of course all sample paths ω. If this were the case, we are done.

2) $\bar{x}^{n_k} \to x^*$ for a subsequence $n_1, n_2, \ldots, n_k, \ldots$. For example, it might be that the sequence $\bar{x}^1, \bar{x}^3, \bar{x}^5, \ldots \to x^*$, while $\mathbb{E}[g^2|\mathfrak{F}^1], \mathbb{E}[g^4|\mathfrak{F}^3], \ldots, \to 0$. This would mean that for the subsequence n_k, $U^{n_k} \to 0$. But we already know that $U^n \to U^*$ where U^* is the unique limit point, which means that $U^* = 0$. But if this is the case, then this is the limit point for every sequence of $\bar{x}^n$.

3) There is no subsequence $\bar{x}^{n_k}$ which has $\bar{x}^*$ as its limit point. This means that $\mathbb{E}[g^{n+1}|\mathfrak{F}^n] \to 0$. However, assumption 3 tells us that the expected gradient cannot vanish at a nonoptimal value of x. This means that this case cannot happen.

This completes the proof. □

5.9 BIBLIOGRAPHIC NOTES

- Section 5.3.3 - See Auer et al. (2002) and Munos & Szepesvári (2008) for examples of finite-time analysis.

PROBLEMS

5.1 In a flexible spending account (FSA), a family is allowed to allocate x pretax dollars to an escrow account maintained by the employer. These funds can be used for medical expenses in the following year. Funds remaining in the account at the end of the following year revert back to the employer. Assume that you are in a 40 percent tax bracket (sounds nice, and the arithmetic is a bit easier). Let M be the random variable representing total medical expenses in the upcoming year, and let $F(x) = Prob[M \leq x]$ be the cumulative distribution function of the random variable M.

a) Write out the objective function that we would want to solve to find x to minimize the total cost (in pretax dollars) of covering your medical expenses next year.

b) If x^* is the optimal solution and $g(x)$ is the gradient of your objective function if you allocate x to the FSA, use the property that $g(x^*) = 0$ to derive (you must show the derivation) the critical ratio that gives the relationship between x^* and the cumulative distribution function $F(x)$.

c) If you are in a 35 percent tax bracket, what percentage of the time should you have funds left over at the end of the year?

5.2 Consider a function $F(x, W)$ that depends on a decision $x = x^n$ after which we observe a random outcome W^{n+1}. Assume that we can compute the gradient $\nabla_x F(x^n, W^{n+1})$. We would like to optimize this problem using a standard stochastic gradient algorithm:

$$x^{n+1} = x^n + \alpha_n \nabla_x F(x^n, W^{n+1}).$$

Our goal is to find the best answer we can after N iterations.

a) Assume that we are using a stepsize policy of

$$\alpha_n = \frac{\theta}{\theta + n - 1}.$$

Model the problem of finding the best stepsize policy as a stochastic optimization problem. Give the state variable(s), the decision variable, the exogenous information, the transition function, and the objective function. Please use precise notation.

b) How does your model change if you switch to the BAKF stepsize rule as implemented in figure 6.8 of the text (page 201 of the latest edition)?

5.3 We are going to solve a classic stochastic optimization problem known as the newsvendor problem. Assume we have to order x assets after which we try to satisfy a random demand D for these assets, where D is randomly distributed between 100 and 200. If $x > D$, we have ordered too much and we pay $5(x - D)$. If $x < D$, we have an underage, and we have to pay $20(D - x)$.

(a) Write down the objective function in the form $\min_x \mathbb{E} f(x, D)$.

(b) Derive the stochastic gradient for this function.

(c) Find the optimal solution analytically [Hint: take the expectation of the stochastic gradient, set it equal to zero and solve for the quantity $\mathbb{P}(D \leq x^*)$. From this, find x^*.]

(d) Since the gradient is in units of dollars while x is in units of the quantity of the asset being ordered, we encounter a scaling problem. Choose as a stepsize $\alpha_{n-1} = \alpha_0/n$ where α_0 is a parameter that has to be chosen. Use $x^0 = 100$ as an initial solution. Plot x^n for 1000 iterations for $\alpha_0 = 1, 5, 10, 20$. Which value of α_0 seems to produce the best behavior?

(e) Repeat the algorithm (1000 iterations) 10 times. Let $\omega = (1, \ldots, 10)$ represent the 10 sample paths for the algorithm, and let $x^n(\omega)$ be the solution at iteration n for sample path ω. Let *Var*(x^n) be the variance of the random variable x^n where

$$\overline{V}(x^n) = \frac{1}{10} \sum_{\omega=1}^{10} (x^n(\omega) - x^*)^2$$

Plot the standard deviation as a function of n for $1 \leq n \leq 1000$.

5.4 A customer is required by her phone company to pay for a minimum number of minutes per month for her cell phone. She pays 12 cents per minute of guaranteed minutes, and 30 cents per minute that she goes over her minimum. Let x be the number of minutes she commits to each month, and let M be the random variable representing the number of minutes she uses each month, where M is normally distributed with mean 300 minutes and a standard deviation of 60 minutes.

(a) Write down the objective function in the form $\min_x \mathbb{E}f(x, M)$.

(b) Derive the stochastic gradient for this function.

(c) Let $x^0 = 0$ and choose as a stepsize $\alpha_{n-1} = 10/n$. Use 100 iterations to determine the optimum number of minutes the customer should commit to each month.

5.5 Show that $\mathbb{E}\left[\left(\bar{\theta}^{n-1} - \theta^n\right)^2\right] = \lambda^{n-1}\sigma^2 + (\beta^n)^2$. [Hint: Add and subtract $\mathbb{E}\bar{\theta}^{n-1}$ inside the expectation and expand.]

5.6 Show that $\mathbb{E}\left[\left(\bar{\theta}^{n-1} - \hat{\theta}^n\right)^2\right] = (1 + \lambda^{n-1})\sigma^2 + (\beta^n)^2$ (which proves equation 3.39). [Hint: See previous exercise.]

5.7 Derive the small sample form of the recursive equation for the variance given in (3.40). Recall that if

$$\bar{\theta}^n = \frac{1}{n}\sum_{m=1}^{n} \hat{\theta}^m$$

then an estimate of the variance of $\hat{\theta}$ is

$$Var[\hat{\theta}] = \frac{1}{n-1}\sum_{m=1}^{n} (\hat{\theta}^m - \bar{\theta}^n)^2.$$

5.8 Consider a random variable given by $R = 10U$ (which would be uniformly distributed between 0 and 10). We wish to use a stochastic gradient algorithm to estimate the mean of R using the iteration $\bar{\theta}^n = \bar{\theta}^{n-1} - \alpha_{n-1}(R^n - \bar{\theta}^{n-1})$, where R^n is a Monte Carlo sample of R in the n^{th} iteration. For each of the stepsize rules below, use equation (5.18) to measure the performance of the stepsize rule to determine which works best, and compute an estimate of the bias and variance at each iteration. If the stepsize rule requires choosing a parameter, justify the choice you make (you may have to perform some test runs).

(a) $\alpha_{n-1} = 1/n$.

(b) Fixed stepsizes of $\alpha_n = .05, .10$ and $.20$.

(c) The stochastic gradient adaptive stepsize rule (equations 6.22)-(6.23)).

(d) The Kalman filter (equations (6.37)-(6.41)).

(e) The optimal stepsize rule (algorithm 6.8).

5.9 Repeat exercise 6.7 using

$$R^n = 10(1 - e^{-0.1n}) + 6(U - 0.5).$$

5.10 Repeat exercise 6.7 using

$$R^n = \left(10/(1+e^{-0.1(50-n)})\right) + 6(U-0.5).$$

5.11 Let U be a uniform $[0,1]$ random variable, and let

$$\mu^n = 1 - \exp\left(-\theta_1 n\right).$$

Now let $\hat{R}^n = \mu^n + \theta_2(U^n - .5)$. We wish to try to estimate μ^n using

$$\bar{R}^n = (1-\alpha_{n-1})\bar{R}^{n-1} + \alpha_{n-1}\hat{R}^n.$$

In the exercises below, estimate the mean (using $\bar{R}^n$) and compute the standard deviation of $\bar{R}^n$ for $n = 1, 2, \ldots, 100$, for each of the following stepsize rules:

- $\alpha_{n-1} = 0.10$.
- $\alpha_{n-1} = a/(a+n-1)$ for $a = 1, 10$.
- Kesten's rule.
- Godfrey's rule.
- The bias-adjusted Kalman filter stepsize rule.

For each of the parameter settings below, compare the rules based on the average error (1) over all 100 iterations and (2) in terms of the standard deviation of $\bar{R}^{100}$.

(a) $\theta_1 = 0, \theta_2 = 10$.

(b) $\theta_1 = 0.05, \theta_2 = 0$.

(c) $\theta_1 = 0.05, \theta_2 = 0.2$.

(d) $\theta_1 = 0.05, \theta_2 = 0.5$.

(e) Now pick the single stepsize that works the best on all four of the above exercises.

5.12 An oil company covers the annual demand for oil using a combination of futures and oil purchased on the spot market. Orders are placed at the end of year $t-1$ for futures that can be exercised to cover demands in year t. If too little oil is purchased this way, the company can cover the remaining demand using the spot market. If too much oil is purchased with futures, then the excess is sold at 70 percent of the spot market price (it is not held to the following year – oil is too valuable and too expensive to store).

To write down the problem, model the exogenous information using

$$\begin{aligned} \hat{D}_t &= \text{Demand for oil during year } t, \\ \hat{p}^s_t &= \text{Spot price paid for oil purchased in year } t, \\ \hat{p}^f_{t,t+1} &= \text{Futures price paid in year } t \text{ for oil to be used in year } t+1. \end{aligned}$$

The demand (in millions of barrels) is normally distributed with mean 600 and standard deviation of 50. The decision variables are given by

$$\begin{aligned} \bar{\theta}^f_{t,t+1} &= \text{Number of futures to be purchased at the end of year } t \text{ to be used in} \\ &\quad\ \text{year } t+1. \\ \bar{\theta}^s_t &= \text{Spot purchases made in year } t. \end{aligned}$$

(a) Set up the objective function to minimize the expected total amount paid for oil to cover demand in a year $t+1$ as a function of $\bar{\theta}_t^f$. List the variables in your expression that are not known when you have to make a decision at time t.

(b) Give an expression for the stochastic gradient of your objective function. That is, what is the derivative of your function for a particular sample realization of demands and prices (in year $t+1$)?

(c) Generate 100 years of random spot and futures prices as follows:

$$\begin{aligned}\hat{p}_t^f &= 0.80 + 0.10 U_t^f,\\ \hat{p}_{t,t+1}^s &= \hat{p}_t^f + 0.20 + 0.10 U_t^s,\end{aligned}$$

where U_t^f and U_t^s are random variables uniformly distributed between 0 and 1. Run 100 iterations of a stochastic gradient algorithm to determine the number of futures to be purchased at the end of each year. Use $\bar{\theta}_0^f = 30$ as your initial order quantity, and use as your stepsize $\alpha_t = 20/t$. Compare your solution after 100 years to your solution after 10 years. Do you think you have a good solution after 10 years of iterating?

5.13 The proof in section 5.8.3 was performed assuming that θ is a scalar. Repeat the proof assuming that θ is a vector. You will need to make adjustments such as replacing Assumption 2 with $\|g^n\| < B$. You will also need to use the triangle inequality which states that $\|a+b\| \leq \|a\| + \|b\|$.

5.14 Prove corollary 6.7.3.

CHAPTER 6

ADAPTIVE ESTIMATION AND STEPSIZE POLICIES

There is a wide range of adaptive learning problems that depend on an iteration of the form we first saw in chapter 5 that looks like

$$x^{n+1} \quad = \quad x^n + \alpha_n \nabla_x F(x^n, W^{n+1}). \tag{6.1}$$

The stochastic gradient $\nabla_x F(x^n, W^{n+1})$ tells us what direction to go in, but we need the stepsize α_n to tell us how far we should move.

There are two important settings where this formula is used. The first is where we are maximizing some metric such as contributions, utility or performance. In these settings, the units of $\nabla_x F(x^{n-1}, W^n)$ and the decision variable x are different, so the stepsize has to perform the scaling so that the size of $\alpha_n \nabla_x F(x^n, W^{n+1})$ is not too large or too small relative to x^n.

A second and very important setting arises in what are known as *supervised learning* settings. In this context, we are trying to estimate some function $f(x|\theta)$ using observations $y = f(x|\theta) + \varepsilon$. In this context, $f(x|\theta)$ and y have the same scale. We encounter these problems in three settings:

- Approximating the function $\mathbb{E}F(x, W)$ to create an estimate $\overline{F}(x)$ that can be optimized.

- Approximating the value $V_t(S_t)$ of being in a state S_t and then following some policy (we encounter this problem starting in chapters 17 and 18 when we introduce approximate dynamic programming).

- Creating a parameterized policy $X^\pi(S|\theta)$ (we might call this $A^\pi(S)$ if we are using action a, or $U^\pi(S)$ if we are using control u). Here, we assume we have access to some method of creating a decision x and then we use this to create a parameterized policy $X^\pi(S|\theta)$.

In chapter 3, we saw a range of methods for approximating functions. Imagine that we face the simplest problem of estimating the mean of a random variable W, which we can show (see exercise 6.5) solves the following stochastic optimization problem

$$\min_x \mathbb{E}\frac{1}{2}(x-W)^2. \tag{6.2}$$

Let $F(x,W) = \frac{1}{2}(x-W)^2$. The stochastic gradient of $F(\mu, W)$ with respect to μ is

$$\nabla_\mu F(x,W) = (x-W).$$

We can optimize (6.2) using a stochastic gradient algorithm which we would write (remember that we are minimizing):

$$\begin{aligned} x^{n+1} &= x^n - \alpha_n \nabla F(x^n, W^{n+1}) && (6.3)\\ &= x^n - \alpha_n(x^n - W^{n+1}) && (6.4)\\ &= (1-\alpha_n)x^n + \alpha_n W^{n+1}. && (6.5) \end{aligned}$$

Equation (6.5) will be familiar to many readers as a smoothing algorithm (also known as a *linear filter* in signal processing). The important observation is that in this setting, the stepsize α_n needs to be between 0 and 1 since μ and W are the same scale.

We made the argument in section 5.3.3 that the iterate in equation (6.1) can be viewed as a dynamical system which is controlled by the stepsize rule, which means it is effectively a form of policy. If we want the best answer within a budget of N iterations, we want to find the best rule (or policy) to achieve this.

We divide our presentation of stepsize rules into three classes:

Deterministic policies - These are stepsize policies that are deterministic functions of the iteration counter n.

Stochastic policies - These are policies where the stepsize at iteration n depends on the statistics computed from the trajectory of the algorithm.

Optimal policies - Our deterministic and stochastic stepsize policies are heuristic, and as a result require tuning one or more parameters. Optimal policies are derived from a formal model which is typically a simplified problem. These policies tend to be more complex, but eliminate or at least minimize the need for parameter tuning.

As we first pointed out in section 3.1.1, the policies we present in this chapter are usually implemented in the computer drawing on the tools of Monte Carlo simulation, which we introduce in more depth in chapter 10. For now, each time we use a sample of a random variable, it is easiest to assume that this is coming from a computer-generated sample, although there are settings where it can come from a field observations.

6.1 DETERMINISTIC STEPSIZE POLICIES

One of the challenges in Monte Carlo methods is finding the stepsize α_n. We refer to a method for choosing a stepsize as a *stepsize policy*, although popular terms include stepsize rule or learning rate schedules. A standard technique in deterministic problems (of the continuously differentiable variety) is to find the value of α_n so that $\bar{\theta}^n$ gives the smallest possible objective function value (among all possible values of α). For a deterministic problem, this is generally not too hard. For a stochastic problem, it means calculating the objective function, which involves computing an expectation. For most applications, expectations are computationally intractable, which makes it impossible to find an optimal stepsize.

Throughout our presentation, we assume that we are using stepsizes to estimate a parameter μ which might be the value $\hat{F} = F(x, W)$ corresponding to a decision x or the value of being in a state s. In chapter 3, we use these techniques to estimate more general parameter vectors when we introduce the idea of using regression models to approximate a value function. We let $\bar{\theta}^{n-1}$ be the estimate of μ after $n-1$ iterations, and we let $\hat{\theta}^n$ be a random observation in iteration n of the value of being in state s ($\hat{\theta}^n$ might be a biased observation of the true value θ^n).

Our updating equation looks like

$$\bar{\theta}^n = (1 - \alpha_{n-1})\bar{\theta}^{n-1} + \alpha_{n-1}\hat{\theta}^n. \tag{6.6}$$

Our iteration counter always starts at $n = 1$ (just as our first time interval starts with $t = 1$). The use of α_{n-1} in equation (6.6) means that we are computing α_{n-1} using information available at iteration $n-1$ and before. Thus, we have an explicit assumption that we are not using $\hat{\theta}^n$ to compute the stepsize in iteration n. This is irrelevant when we use a deterministic stepsize sequence, but is critical in convergence proofs for stochastic stepsize formulas (introduced below). In most formulas, α_0 is a parameter that has to be specified, although we will generally assume that $\alpha_0 = 1$, which means that we do not have to specify $\bar{\theta}_0$. The only reason to use $\alpha_0 < 1$ is when we have some *a priori* estimate of $\bar{\theta}_0$ which is better than $\hat{\theta}^1$.

There are two issues when designing a good stepsize rule. The first is the question of whether the stepsize produces some theoretical guarantee, such as asymptotic convergence or a finite time bound. While this is primarily of theoretical interest, these conditions do provide important guidelines to follow to produce good behavior. The second issue is whether the rule produces good empirical performance.

Below, we start with a general discussion of stepsize rules. Following this, we provide a number of examples of deterministic stepsize rules. These are formulas that depend only on the iteration counter n (or more precisely, the number of times that we update a particular parameter). Section 6.2 then describes stochastic stepsize rules that adapt to the data.

The deterministic and stochastic rules presented in this section and section 6.2 are, for the most part, heuristically designed to achieve good rates of convergence, but are not supported by any theory that they will produce the best rate of convergence. Later (section 6.3) we provide a theory for choosing stepsizes that produce the fastest possible rate of convergence when estimating value functions based on policy evaluation. Finally, section 6.4 presents a new optimal stepsize rule designed specifically for approximate value iteration.

6.1.1 Properties for convergence

The theory for proving convergence of stochastic gradient algorithms was first developed in the early 1950's and has matured considerably since then (see section 5.8). However, all the proofs require three basic conditions

$$\alpha_{n-1} > 0, \quad n = 1, 2, \ldots, \tag{6.7}$$

$$\sum_{n=1}^{\infty} \alpha_{n-1} = \infty, \tag{6.8}$$

$$\sum_{n=1}^{\infty} (\alpha_{n-1})^2 < \infty. \tag{6.9}$$

Equation (6.7) obviously requires that the stepsizes be strictly positive (we cannot allow stepsizes equal to zero). The most important requirement is (6.8), which states that the infinite sum of stepsizes must be infinite. If this condition did not hold, the algorithm might stall prematurely. Finally, condition 6.9 requires that the infinite sum of the squares of the stepsizes be finite. This condition, in effect, requires that the stepsize sequence converge "reasonably quickly." A good intuitive justification for this condition is that it guarantees that the *variance* of our estimate of the optimal solution goes to zero in the limit. Sections 5.8.2 and 5.8.3 illustrate two proof techniques that both lead to these requirements on the stepsize. However, it is possible under certain conditions to replace equation (6.9) with the weaker requirement that $\lim_{n\rightarrow\infty} \alpha_n = 0$.

Conditions (6.8) and (6.9) effectively require that the stepsizes decline according to an arithmetic sequence such as

$$\alpha_{n-1} = \frac{1}{n}. \tag{6.10}$$

This rule has an interesting property. Exercise 6.5 asks you to show that a stepsize of $1/n$ produces an estimate $\bar{\theta}^n$ that is simply an average of all previous observations, which is to say

$$\bar{\theta}^n = \frac{1}{n} \sum_{m=1}^{n} \hat{\theta}^m. \tag{6.11}$$

Of course, we have a nice name for equation (6.11): it is called a sample average. And we are all aware that in general (some modest technical conditions are required) as $n \rightarrow \infty$, $\bar{\theta}^n$ will converge (in some sense) to the mean of our random variable R.

The issue of the rate at which the stepsizes decrease is of considerable practical importance. Consider, for example, the stepsize sequence

$$\alpha_n = .5\alpha_{n-1},$$

which is a geometrically decreasing progression. This stepsize formula violates (6.8). More intuitively, the problem is that the stepsizes would decrease so quickly that it is likely that we would never reach the final solution.

There are settings where the "$1/n$" stepsize formula is the best that we can do (as in finding the mean of a random variable), while in other situations it can perform extremely poorly because it can decline to zero too quickly. The situations where it works poorly arise when we are estimating a function that is changing over time (or iterations). For example, there is an algorithmic strategy called Q-learning which involves two steps:

$$\begin{aligned}
\hat{q}^n(s^n, a^n) &= r(s^n, a^n) + \gamma \max_{a'} \bar{Q}^{n-1}(s', a'), \\
\bar{Q}^n(s^n, a^n) &= (1 - \alpha_{n-1})\bar{Q}^{n-1}(s^n, a^n) + \alpha_{n-1}\hat{q}^n(s^n, a^n).
\end{aligned}$$

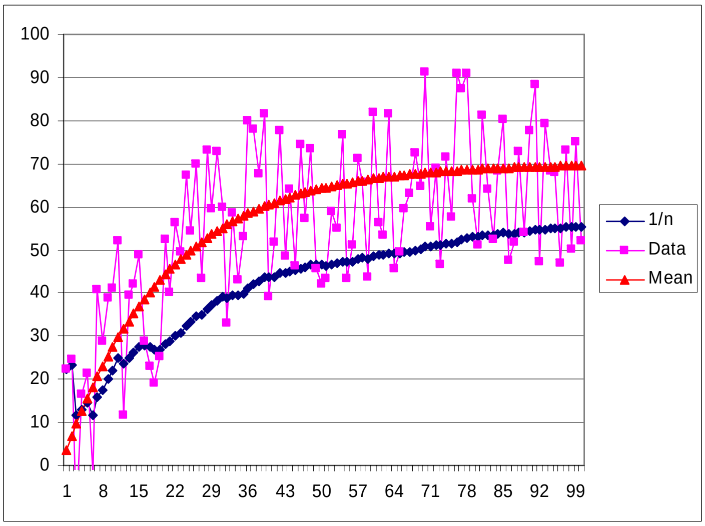


Figure 6.1 Illustration of poor convergence of the $1/n$ stepsize rule in the presence of transient data.

Here, we create a sampled observation $\hat{q}^n(s^n, a^n)$ of being in a state s^n and taking an action a^n, which we compute using the one period reward $r(s^n, a^n)$ plus an estimate of the downstream value, computed by sampling a downstream state s' given the current state s^n and action a^n, and then choosing the best action a' based on our current estimate of the value of different state-action pairs $\bar{Q}^{n-1}(s', a')$. We then smooth $\hat{q}^n(s^n, a^n)$ using our stepsize α_{n-1} to obtain updated estimates $\bar{Q}^n(s^n, a^n)$ of the value of the state-action pair s^n and a^n.

Figure 6.1 illustrates the behavior of using $1/n$ in this setting, which shows that we are significantly underestimating the values. Below, we fix this by generalizing $1/n$ using a tunable parameter. Later, we are going to present stepsize formulas that help to mitigate this behavior.

6.1.2 Deterministic stepsize policies

The remainder of this section presents a series of deterministic stepsize formulas designed to overcome this problem. These rules are the simplest to implement and are typically a good starting point when implementing adaptive learning algorithms.

Constant stepsizes

A constant stepsize rule is simply

$$\alpha_{n-1} = \begin{cases} 1 & \text{if } n = 1, \\ \bar{\alpha} & \text{otherwise,} \end{cases}$$

where $\bar{\alpha}$ is a stepsize that we have chosen. It is common to start with a stepsize of 1 so that we do not need an initial value $\bar{\theta}^0$ for our statistic.

Constant stepsizes are popular when we are estimating not one but many parameters (for large scale applications, these can easily number in the thousands or millions). In these cases, no single rule is going to be right for all of the parameters and there is enough noise that any reasonable stepsize rule will work well. Constant stepsizes are easy to code (no memory requirements) and, in particular, easy to tune (there is only one parameter). Perhaps the biggest point in their favor is that we simply may not know the rate of convergence, which means that we run the risk with a declining stepsize rule of allowing the stepsize to decline too quickly, producing a behavior we refer to as "apparent convergence."

In dynamic programming, we are typically trying to estimate the value of being in a state using observations that are not only random, but which are also changing systematically as we try to find the best policy. As a general rule, as the noise in the observations of the values increases, the best stepsize decreases. But if the values are increasing rapidly, we want a larger stepsize. Choosing the best stepsize requires striking a balance between stabilizing the noise and responding to the changing mean. Figure 6.2 illustrates observations that are coming from a process with relatively low noise but where the mean is changing quickly (6.2a), and observations that are very noisy but where the mean is not changing at all (6.2b). For the first, the ideal stepsize is relatively large, while for the second, the best stepsize is quite small.

Generalized harmonic stepsizes

A generalization of the $1/n$ rule is the generalized harmonic sequence given by

$$\alpha_{n-1} = \frac{\theta}{\theta + n - 1}. \tag{6.12}$$

This rule satisfies the conditions for convergence, but produces larger stepsizes for $\theta > 1$ than the $1/n$ rule. Increasing a slows the rate at which the stepsize drops to zero, as illustrated in figure 6.3. In practice, it seems that despite theoretical convergence proofs to the contrary, the stepsize $1/n$ can decrease to zero far too quickly, resulting in "apparent convergence" when in fact the solution is far from the best that can be obtained.

Polynomial learning rates

An extension of the basic harmonic sequence is the stepsize

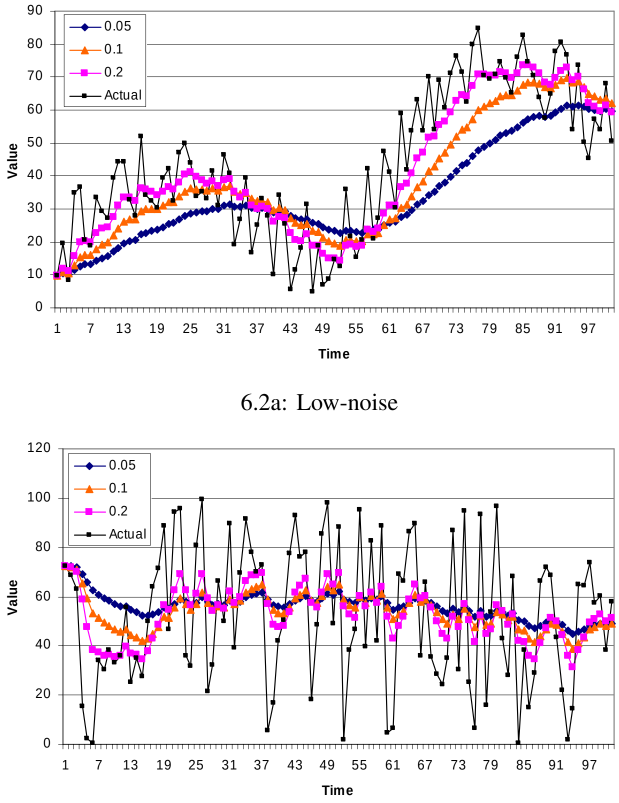
$$\alpha_{n-1} = \frac{1}{(n)^\beta}, \tag{6.13}$$

where $\beta \in (\frac{1}{2}, 1]$. Smaller values of β slow the rate at which the stepsizes decline, which improves the responsiveness in the presence of initial transient conditions. The best value of β depends on the degree to which the initial data is transient, and as such is a parameter that needs to be tuned.

McClain's formula

McClain's formula is an elegant way of obtaining $1/n$ behavior initially but approaching a specified constant in the limit. The formula is given by

$$\alpha_n = \frac{\alpha_{n-1}}{1 + \alpha_{n-1} - \bar{\alpha}}. \tag{6.14}$$



6.2a: Low-noise

6.2b: High-noise

Figure 6.2 Illustration of the effects of smoothing using constant stepsizes. Case (a) represents a low-noise dataset, with an underlying nonstationary structure; case (b) is a high-noise dataset from a stationary process.

where $\bar{\alpha}$ is a specified parameter. Note that steps generated by this model satisfy the following properties

$$\begin{aligned} \alpha_n > \alpha_{n+1} > \bar{\alpha} \quad &\text{if} \quad \alpha > \bar{\alpha}, \\ \alpha_n < \alpha_{n+1} < \bar{\alpha} \quad &\text{if} \quad \alpha < \bar{\alpha}. \end{aligned}$$

McClain's rule, illustrated in figure 6.4, combines the features of the "$1/n$" rule which is ideal for stationary data, and constant stepsizes for nonstationary data. If we set $\bar{\alpha} = 0$, then it is easy to verify that McClain's rule produces $\alpha_{n-1} = 1/n$. In the limit, $\alpha_n \rightarrow \bar{\alpha}$. The value of the rule is that the $1/n$ averaging generally works quite well in the very first iterations (this is a major weakness of constant stepsize rules), but avoids going to zero. The rule can be effective when you are not sure how many iterations are required to start converging, and it can also work well in nonstationary environments.

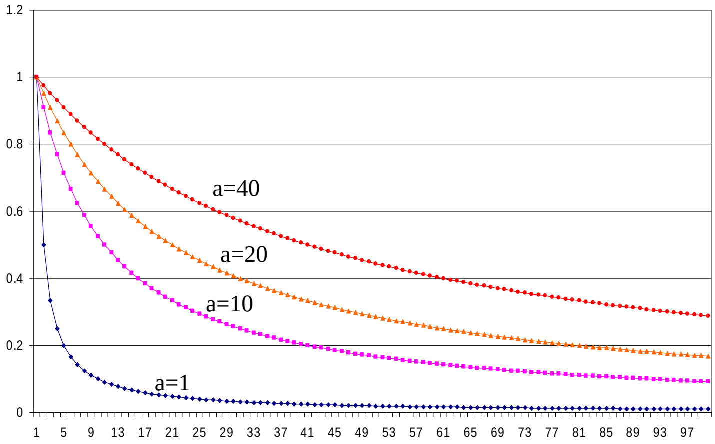


Figure 6.3 Stepsizes for $a/(a+n)$ while varying a.

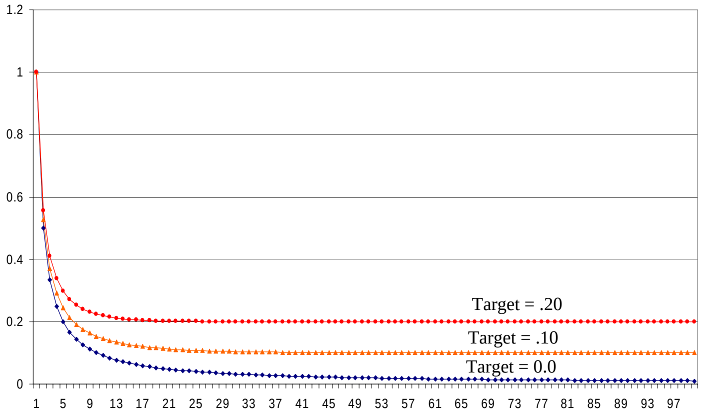


Figure 6.4 The McClain stepsize rule with varying targets.

Search-then-converge learning rule

The search-then-converge (STC) stepsize rule is a variation on the harmonic stepsize rule that produces delayed learning. The rule can be written as

$$\alpha_{n-1} = \alpha_0 \frac{\left(\frac{b}{n} + a\right)}{\left(\frac{b}{n} + a + n^{\beta}\right)}. \tag{6.15}$$

If $\beta = 1$, then this formula is similar to the STC rule. In addition, if $b = 0$, then it is the same as the $a/(a+n)$ rule. The addition of the term b/n to the numerator and the denominator can be viewed as a kind of $a/(a+n)$ rule where a is very large but declines with n. The effect of the b/n term, then, is to keep the stepsize larger for a longer period of time, as illustrated in figure 6.5. This can help algorithms that have to go through an extended learning phase when the values being estimated are relatively unstable. The

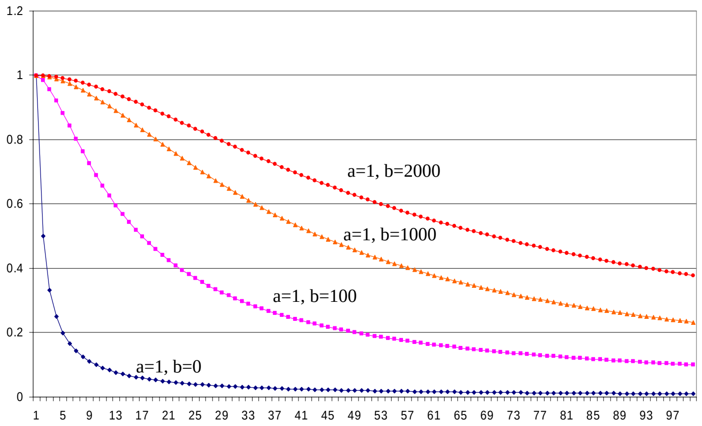


Figure 6.5 Stepsizes for $(b/n + a)/(b/n + a + n)$ while varying b.

relative magnitude of b depends on the number of iterations which are expected to be run, which can range from several dozen to several million.

This class of stepsize rules is termed "search-then-converge" because they provide for a period of high stepsizes (while searching is taking place) after which the stepsize declines (to achieve convergence). The degree of delayed learning is controlled by the parameter b, which can be viewed as playing the same role as the parameter a but which declines as the algorithm progresses. The rule is designed for approximate dynamic programming methods applied to the setting of playing games with a delayed reward (there is no reward until you win or lose the game).

The exponent β in the denominator has the effect of increasing the stepsize in later iterations (see figure 6.6). With this parameter, it is possible to accelerate the reduction of the stepsize in the early iterations (by using a smaller a) but then slow the descent in later iterations (to sustain the learning process). This may be useful for problems where there is an extended transient phase requiring a larger stepsize for a larger number of iterations.

6.2 STOCHASTIC STEPSIZE POLICIES

There is considerable appeal to the idea that the stepsize should depend on the actual trajectory of the algorithm. For example, if we are consistently observing that our estimate $\bar{\theta}^{n-1}$ is smaller (or larger) than the observations $\hat{\theta}^n$, then it suggests that we are trending upward (or downward). When this happens, we typically would like to use a larger stepsize to increase the speed at which we reach a good estimate. When the stepsizes depend on the observations $\hat{\theta}^n$, then we say that we are using a *stochastic stepsize*.

In this section, we first review the case for stochastic stepsizes, then present the revised theoretical conditions for convergence, and finally outline a series of heuristic recipes that have been suggested in the literature. After this, we present some stepsize rules that are optimal until special conditions.

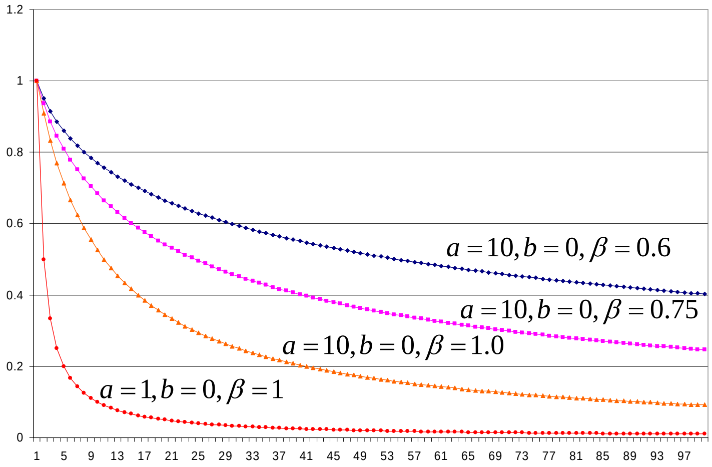


Figure 6.6 Stepsizes for $(b/n + a)/(b/n + a + n^{\beta})$ while varying β.

6.2.1 The case for stochastic stepsizes

Assume that our estimates are consistently under or consistently over the actual observations. This can easily happen during early iterations due to either a poor initial starting point or the use of biased estimates (which is common in dynamic programming) during the early iterations. For large problems, it is possible that we have to estimate thousands of parameters. It seems unlikely that all the parameters will approach their true value at the same rate. Figure 6.7 shows the change in estimates of the value of being in different states, illustrating the wide variation in learning rates that can occur within the same dynamic program.

Stochastic stepsizes try to adjust to the data in a way that keeps the stepsize larger while the parameter being estimated is still changing quickly. Balancing noise against the change in the underlying signal, particularly when both of these are unknown, is a difficult challenge.

6.2.2 Convergence conditions

When the stepsize depends on the history of the process, the stepsize itself becomes a random variable. This change requires some subtle modifications to our requirements for convergence (equations (6.8) and (6.9)). For technical reasons, our convergence criteria change to

$$\begin{aligned} \alpha_n &> 0, \text{ almost surely,} && (6.16) \\ \sum_{n=0}^{\infty} \alpha_n &= \infty \text{ almost surely,} && (6.17) \\ \mathbb{E}\left\{ \sum_{n=0}^{\infty} (\alpha_n)^2 \right\} &< \infty. && (6.18) \end{aligned}$$

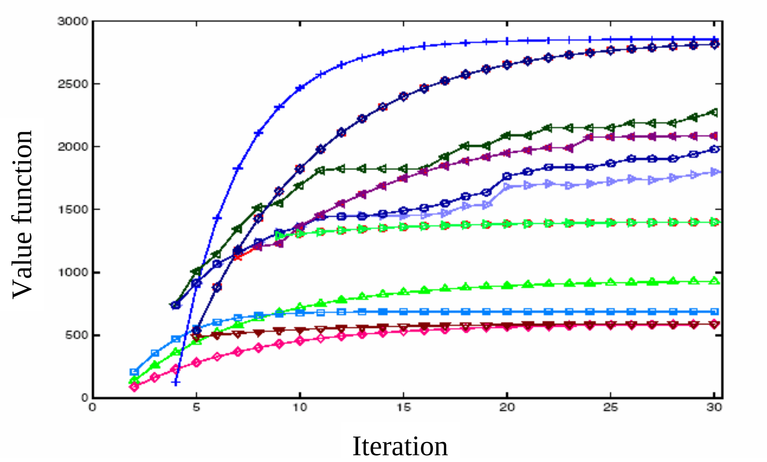


Figure 6.7 Different parameters can undergo significantly different initial rates.

The condition "almost surely" (universally abbreviated "a.s.") means that equation (6.17) holds for every sample path ω, and not just on average. More precisely, we mean every sample path ω that might actually happen (we exclude sample paths where the probability that the sample path would happen is zero).

For the reasons behind these conditions, go to our "Why does it work" section (5.8). It is important to emphasize, however, that these conditions are completely unverifiable and are purely for theoretical reasons. The real issue with stochastic stepsizes is whether they contribute to the rate of convergence.

6.2.3 A selection of policies

The desire to find stepsize policies that adapt to the data has become a small cottage industry which has produced a variety of formulas with varying degrees of sophistication and convergence guarantees. This section provides a brief sample of some popular policies, some (such as AdaGrad) with strong performance guarantees. Later, we present some optimal policies for specialized problems.

To present our stochastic stepsize formulas, we need to define a few quantities. Recall that our basic updating expression is given by

$$\bar{\theta}^n \quad = \quad (1-\alpha_{n-1})\bar{\theta}^{n-1} + \alpha_{n-1}\hat{\theta}^n.$$

$\bar{\theta}^{n-1}$ is our estimate of the next observation, given by $\hat{\theta}^n$. The difference between the estimate and the actual can be treated as the error, given by

$$\varepsilon^n = \bar{\theta}^{n-1} - \hat{\theta}^n.$$

We may wish to smooth the error in the estimate, which we designate by the function

$$S(\varepsilon^n) = (1-\beta)S(\varepsilon^{n-1}) + \beta\varepsilon^n.$$

Some formulas depend on tracking changes in the sign of the error. This can be done using the indicator function

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

Thus, $1_{\varepsilon^n \varepsilon^{n-1} < 0}$ indicates if the sign of the error has changed in the last iteration.

Below, we summarize three classic rules. Kesten's rule is the oldest and is perhaps the simplest illustration of a stochastic stepsize rule. Trigg's formula is a simple rule widely used in the demand forecasting community. Finally, the stochastic gradient adaptive stepsize rule enjoys a theoretical convergence proof, but is controlled by several tunable parameters that complicate its use in practice.

Kesten's rule

Kesten's rule was one of the earliest stepsize rules which took advantage of a simple principle. If we are far from the optimal, the errors tend to all have the same sign. As we get close, the errors tend to alternate. Exploiting this simple observation, Kesten proposed the following simple rule:

$$\alpha_{n-1} = \frac{a}{a + K^n - 1}, \tag{6.19}$$

where a is a parameter to be calibrated. K^n counts the number of times that the sign of the error has changed, where we use

$$K^n = \begin{cases} n & \text{if } n = 1, 2, \\ K^{n-1} + 1_{\{\varepsilon^n \varepsilon^{n-1} < 0\}} & \text{if } n > 2. \end{cases} \tag{6.20}$$

Kesten's rule is particularly well suited to initialization problems. It slows the reduction in the stepsize as long as the error exhibits the same sign (and indication that the algorithm is still climbing into the correct region). However, the stepsize declines monotonically. This is typically fine for most dynamic programming applications, but can encounter problems in situations with delayed learning.

Trigg's formula

Trigg's formula is given by

$$\alpha_n = \frac{|S(\varepsilon^n)|}{S(|\varepsilon^n|)}. \tag{6.21}$$

The formula takes advantage of the simple property that smoothing on the absolute value of the errors is greater than or equal to the absolute value of the smoothed errors. If there is a series of errors with the same sign, that can be taken as an indication that there is a significant difference between the true mean and our estimate of the mean, which means we would like larger stepsizes.

Stochastic gradient adaptive stepsize rule

This class of rules uses stochastic gradient logic to update the stepsize. We first compute

$$\psi^n = (1 - \alpha_{n-1})\psi^{n-1} + \varepsilon^n. \tag{6.22}$$

The stepsize is then given by

$$\alpha_n = \left[\alpha_{n-1} + \nu\psi^{n-1}\varepsilon^n\right]_{\alpha_-}^{\alpha_+}, \tag{6.23}$$

where α_+ and α_- are, respectively, upper and lower limits on the stepsize. $[\cdot]_{\alpha_-}^{\alpha_+}$ represents a projection back into the interval $[\alpha_-, \alpha_+]$, and ν is a scaling factor. $\psi^{n-1}\varepsilon^n$ is a stochastic gradient that indicates how we should change the stepsize to improve the error. Since the stochastic gradient has units that are the square of the units of the error, while the stepsize is unitless, ν has to perform an important scaling function. The equation $\alpha_{n-1} + \nu\psi^{n-1}\varepsilon^n$ can easily produce stepsizes that are larger than 1 or smaller than 0, so it is customary to specify an allowable interval (which is generally smaller than (0,1)). This rule has provable convergence, but in practice, ν, α_+ and α_- all have to be tuned.

ADAM ADAM (Adaptive Moment Estimation) is another stepsize policy that has attracted attention in recent years. As above, let $g^n = \nabla_x F(x^{n-1}, W^n)$ be our gradient, and let g_i^n be the ith element. ADAM proceeds by adaptively computing means and variances according to

$$m_i^n = \beta_1 m_i^{n-1} + (1-\beta_1)g_i^n, \tag{6.24}$$

$$v_i^n = \beta_2 v_i^{n-1} + (1-\beta_2)(g_i^n)^2. \tag{6.25}$$

These updating equations introduce biases when the data is nonstationary, which is typically the case in stochastic optimization. ADAM compensates for these biases using

$$\begin{aligned} \bar{m}_i^n &= \frac{m_i^n}{1-\beta_1}, \\ \bar{v}_i^n &= \frac{v_i^n}{1-\beta_2}. \end{aligned}$$

The stochastic gradient equation for ADAM is then given by

$$x_i^{n+1} = x_i^n + \frac{\eta}{\sqrt{\bar{v}_i^n} + \epsilon}\bar{m}_i^n. \tag{6.26}$$

AdaGrad AdaGrad ("adaptive gradient") is a relatively recent stepsize policy that has attracted considerable attention in the machine learning literature which not only enjoys nice theoretical performance guarantees, but has also become quite popular because it seems to work quite well in practice.

Assume that we are trying to solve our standard problem

$$\max_x \mathbb{E}_W F(x, W),$$

where we make the assumption that not only is x a vector, but also that the scaling for each dimension might be different (an issue we have ignored so far). To simplify the notation a bit, let the stochastic gradient with respect to $x_i,\ i = 1, \ldots, I$ be given by

$$g_i^n = \nabla_{x_i} F(x^{n-1}, W^n).$$

Now create a $I \times I$ diagonal matrix G^n where the $(i,i)th$ element G_{ii}^n is given by

$$G_{ii}^n = \sum_{m=1}^n (g_i^n)^2.$$

We then set a stepsize for the ith dimension using

$$\alpha_{ni} = \frac{\eta}{(G^n_{ii})^2 + \epsilon}, \tag{6.27}$$

where ϵ is a small number (e.g. 10^{-8} to avoid the possibility of dividing by zero). This can be written in matrix form using

$$\alpha_n = \frac{\eta}{\sqrt{G^n + \epsilon}} \odot g_t, \tag{6.28}$$

where α_n is an I-dimensional matrix. The right way to understand equation (6.28) is equation (6.27).

AdaGrad does an unusually good job of adapting to the behavior of a function. It also adapts to potentially different behaviors of each dimension. For example, we might be solving a machine learning problem to learn a parameter vector θ (this would be the decision variable instead of x) for a linear model of the form

$$y = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + \ldots.$$

The explanatory variables $X_1, X_2, \ldots$ can take on values in completely different ranges. In a medical setting, X_1 might be blood sugar with values between 5 and 8, while X_2 might be the weight of a patient that could range between 100 and 300 pounds. The coefficients θ_1 and θ_2 would be scaled according to the inverse of the scales of the explanatory variables.

6.2.4 Experimental notes

Throughout our presentation, we represent the stepsize at iteration n using α_{n-1}. For discrete, lookup-table representations of value functions (as we are doing here), the stepsize should reflect how many times we have visited a specific state. If $n(S)$ is the number of times we have visited state S, then the stepsize for updating $\overline{V}(S)$ should be $\alpha_{n(S)}$. For notational simplicity, we suppress this capability, but it can have a significant impact on the empirical rate of convergence.

A word of caution is offered when testing out stepsize rules. It is quite easy to test out these ideas in a controlled way in a simple spreadsheet on randomly generated data, but there is a big gap between showing a stepsize that works well in a spreadsheet and one that works well in specific applications. Stochastic stepsize rules work best in the presence of transient data where the degree of noise is not too large compared to the change in the signal (the mean). As the variance of the data increases, stochastic stepsize rules begin to suffer and simpler (deterministic) rules tend to work better.

6.3 OPTIMAL STEPSIZE POLICIES

Given the variety of stepsize formulas we can choose from, it seems natural to ask whether there is an optimal stepsize rule. Before we can answer such a question, we have to define exactly what we mean by it. Assume that we are trying to estimate a parameter (such as a value of being in a state or the slope of a value function) that we denote by θ^n that may be changing over time. At iteration n, our estimate of θ^n, $\bar{\theta}^n$, is a random variable that depends on our stepsize rule. To express this dependence, let α represent a stepsize rule, and let $\bar{\theta}^n(\alpha)$ be the estimate of the parameter μ after iteration n using stepsize rule α. We would like to choose a stepsize rule to minimize

$$\min_{\alpha} \mathbb{E}(\bar{\theta}^n(\alpha) - \theta^n)^2. \tag{6.29}$$

Here, the expectation is over the entire history of the algorithm and requires (in principle) knowing the true value of the parameter being estimated. If we could solve this problem (which requires knowing certain parameters about the underlying distributions), we would obtain a deterministic stepsize rule. In practice, we do not generally know these parameters which need to be estimated from data, producing a stochastic stepsize rule.

There are other objective functions we could use. For example, instead of minimizing the distance to an unknown parameter sequence θ^n, we could solve the minimization problem

$$\min_{\alpha} \mathbb{E}\left\{(\bar{\theta}^n(\alpha) - \hat{\theta}^{n+1})^2\right\}, \tag{6.30}$$

where we are trying to minimize the deviation between our prediction, obtained at iteration n, and the actual observation at $n+1$. Here, we are again proposing an unconditional expectation, which means that $\bar{\theta}^n(\alpha)$ is a random variable within the expectation. Alternatively, we could condition on our history up to iteration n

$$\min_{\alpha} \mathbb{E}^n \left\{(\bar{\theta}^n(\alpha) - \hat{\theta}^{n+1})^2\right\} \tag{6.31}$$

where $\mathbb{E}^n$ means that we are taking the expectation given what we know at iteration n (which means that $\bar{\theta}^n(\alpha)$ is a constant). For readers familiar with the language of filtrations, we would write the expectation as $\mathbb{E}\left\{(\bar{\theta}^n(\alpha) - \hat{\theta}^{n+1})^2|H^n\right\}$, where H^n is the history of the process up through iteration n (that is, the entire sequence $W^1, \ldots, W^n$). In this formulation $\bar{\theta}^n(\alpha)$ is now deterministic at iteration n (because we are conditioning on the history up through iteration n), whereas in (6.30), $\bar{\theta}^n(\alpha)$ is random (since we are not conditioning on the history). The difference between these two objective functions is subtle but significant.

We begin our discussion of optimal stepsizes in section 6.3.1 by addressing the case of estimating a constant parameter which we observe with noise. Section 6.3.2 considers the case where we are estimating a parameter that is changing over time, but where the changes have mean zero. Finally, section 6.3.3 addresses the case where the mean may be drifting up or down with nonzero mean, a situation that we typically face when approximating a value function.

6.3.1 Optimal stepsizes for stationary data

Assume that we observe $\hat{\theta}^n$ at iteration n and that the observations $\hat{\theta}^n$ can be described by

$$\hat{\theta}^n = \theta + \varepsilon^n$$

where μ is an unknown constant and ε^n is a stationary sequence of independent and identically distributed random deviations with mean 0 and variance σ^2. We can approach the problem of estimating μ from two perspectives: choosing the best stepsize and choosing the best linear combination of the estimates. That is, we may choose to write our estimate $\bar{\theta}^n$ after n observations in the form

$$\bar{\theta}^n = \sum_{m=1}^{n} a_m^n \hat{\theta}_m.$$

For our discussion, we will fix n and work to determine the coefficients a_m (recognizing that they can depend on the iteration). We would like our statistic to have two properties: It should be unbiased, and it should have minimum variance (that is, it should solve (6.29)). To be unbiased, it should satisfy

$$\begin{aligned}\mathbb{E}\left[\sum_{m=1}^{n} a_m\hat{\theta}_m\right] &= \sum_{m=1}^{n} a_m\mathbb{E}\hat{\theta}_m \\ &= \sum_{m=1}^{n} a_m\theta \\ &= \theta,\end{aligned}$$

which implies that we must satisfy

$$\sum_{m=1}^{n} a_m = 1.$$

The variance of our estimator is given by:

$$\mathit{Var}(\bar{\theta}^n) = \mathit{Var}\left[\sum_{m=1}^{n} a_m\hat{\theta}_m\right].$$

We use our assumption that the random deviations are independent, which allows us to write

$$\begin{aligned}\mathit{Var}(\bar{\theta}^n) &= \sum_{m=1}^{n} \mathit{Var}[a_m\hat{\theta}_m] \\ &= \sum_{m=1}^{n} a_m^2\mathit{Var}[\hat{\theta}_m] \\ &= \sigma^2\sum_{m=1}^{n} a_m^2. \qquad (6.32)\end{aligned}$$

Now we face the problem of finding $a_1, \ldots, a_n$ to minimize (6.32) subject to the requirement that $\sum_m a_m = 1$. This problem is easily solved using the Lagrange multiplier method. We start with the nonlinear programming problem

$$\min_{\{a_m\}} \sum_{m=1}^{n} a_m^2$$

subject to

$$\begin{aligned}\sum_{m=1}^{n} a_m &= 1, \qquad (6.33)\\ a_m &\geq 0. \qquad (6.34)\end{aligned}$$

We relax constraint (6.33) and add it to the objective function

$$\min_{\{a_m\}} L(a,\lambda) = \sum_{m=1}^{n} a_m^2 - \lambda\left(\sum_{m=1}^{n} a_m - 1\right)$$

subject to (6.34). We are now going to try to solve $L(a, \lambda)$ (known as the "Lagrangian") and hope that the coefficients a are all nonnegative. If this is true, we can take derivatives and set them equal to zero

$$\frac{\partial L(a, \lambda)}{\partial a_m} = 2a_m - \lambda. \tag{6.35}$$

The optimal solution (a^*, λ^*) would then satisfy

$$\frac{\partial L(a, \lambda)}{\partial a_m} = 0.$$

This means that at optimality

$$a_m = \lambda/2,$$

which tells us that the coefficients a_m are all equal. Combining this result with the requirement that they sum to one gives the expected result:

$$a_m = \frac{1}{n}.$$

In other words, our best estimate is a sample average. From this (somewhat obvious) result, we can obtain the optimal stepsize, since we already know that $\alpha_{n-1} = 1/n$ is the same as using a sample average.

This result tells us that if the underlying data is stationary, and we have no prior information about the sample mean, then the best stepsize rule is the basic $1/n$ rule. Using any other rule requires that there be some violation in our basic assumptions. In practice, the most common violation is that the observations are not stationary because they are derived from a process where we are searching for the best solution.

6.3.2 Optimal stepsizes for nonstationary data - I

Assume now that our parameter evolves over time (iterations) according to the process

$$\theta^n = \theta^{n-1} + \xi^n, \tag{6.36}$$

where $\mathbb{E}\xi^n = 0$ is a zero mean drift term with variance $(\sigma^\xi)^2$. As before, we measure θ^n with an error according to

$$\hat{\theta}^n = \theta^n + \varepsilon^n.$$

We want to choose a stepsize so that we minimize the mean squared error. This problem can be solved using the Kalman filter. The Kalman filter is a powerful recursive regression technique, but we adapt it here for the problem of estimating a single parameter. Typical applications of the Kalman filter assume that the variance of ξ^n, given by $(\sigma^\xi)^2$, and the variance of the measurement error, ε^n, given by σ^2, are known. In this case, the Kalman filter would compute a stepsize (generally referred to as the gain) using

$$\alpha_n = \frac{(\sigma^\xi)^2}{\nu^n + \sigma^2}, \tag{6.37}$$

where ν^n is computed recursively using

$$\nu^n = (1 - \alpha_{n-1})\nu^{n-1} + (\sigma^\xi)^2. \tag{6.38}$$

Remember that $\alpha_0 = 1$, so we do not need a value of ν^0. For our application, we do not know the variances so these have to be estimated from data. We first estimate the bias using

$$\bar{\beta}^n \quad = \quad (1 - \eta_{n-1})\,\bar{\beta}^{n-1} + \eta_{n-1}\left(\bar{\theta}^{n-1} - \hat{\theta}^n\right), \tag{6.39}$$

where η_{n-1} is a simple stepsize rule such as the harmonic stepsize rule or McClain's formula. We then estimate the total error sum of squares using

$$\bar{\nu}^n \quad = \quad (1 - \eta_{n-1})\,\bar{\nu}^{n-1} + \eta_{n-1}\left(\bar{\theta}^{n-1} - \hat{\theta}^n\right)^2. \tag{6.40}$$

Finally, we estimate the variance of the error using

$$(\bar{\sigma}^n)^2 \quad = \quad \frac{\bar{\nu}^n - (\bar{\beta}^n)^2}{1 + \bar{\lambda}^{n-1}}, \tag{6.41}$$

where $\bar{\lambda}^{n-1}$ is computed using

$$\lambda^n = \begin{cases} (\alpha_{n-1})^2, & n = 1, \\ (1 - \alpha_{n-1})^2 \lambda^{n-1} + (\alpha_{n-1})^2, & n > 1. \end{cases}$$

We use $(\bar{\sigma}^n)^2$ as our estimate of σ^2. We then propose to use $\left(\bar{\beta}^n\right)^2$ as our estimate of $(\sigma^\xi)^2$. This is purely an approximation, but experimental work suggests that it performs quite well, and it is relatively easy to implement.

6.3.3 Optimal stepsizes for nonstationary data - II

In dynamic programming, we are trying to estimate the value of being in a state (call it v) by $\bar{v}$ which is estimated from a sequence of random observations $\hat{v}$. The problem we encounter is that $\hat{v}$ might depend on a value function approximation which is steadily increasing, which means that the observations $\hat{v}$ are nonstationary. Furthermore, unlike the assumption made by the Kalman filter that the mean of $\hat{v}$ is varying in a zero-mean way, our observations of $\hat{v}$ might be steadily increasing. This would be the same as assuming that $\mathbb{E}\xi = \mu > 0$ in the section above. In this section, we derive the Kalman filter learning rate for biased estimates.

Our challenge is to devise a stepsize that strikes a balance between minimizing error (which prefers a smaller stepsize) and responding to the nonstationary data (which works better with a large stepsize). We return to our basic model

$$\hat{\theta}^n = \theta^n + \varepsilon^n,$$

where θ^n varies over time, but it might be steadily increasing or decreasing. This would be similar to the model in the previous section (equation (6.36)) but where ξ^n has a nonzero mean. As before we assume that $\{\varepsilon^n\}_{n=1,2,\ldots}$ are independent and identically distributed with mean value of zero and variance, σ^2. We perform the usual stochastic gradient update to obtain our estimates of the mean

$$\bar{\theta}^n(\alpha_{n-1}) \quad = \quad (1 - \alpha_{n-1})\,\bar{\theta}^{n-1} + \alpha_{n-1}\hat{\theta}^n. \tag{6.42}$$

We wish to find α_{n-1} that solves,

$$\min_{\alpha_{n-1}} F(\alpha_{n-1}) = \mathbb{E}\left[\left(\bar{\theta}^n(\alpha_{n-1}) - \theta^n\right)^2\right]. \tag{6.43}$$

It is important to realize that we are trying to choose α_{n-1} to minimize the *unconditional* expectation of the error between $\bar{\theta}^n$ and the true value θ^n. For this reason, our stepsize rule will be deterministic, since we are not allowing it to depend on the information obtained up through iteration n.

We assume that the observation at iteration n is unbiased, which is to say

$$\mathbb{E}\left[\hat{\theta}^n\right] \quad = \quad \theta^n. \tag{6.44}$$

But the smoothed estimate is biased because we are using simple smoothing on nonstationary data. We denote this bias as

$$\begin{aligned} \beta^{n-1} &= \mathbb{E}\left[\bar{\theta}^{n-1} - \theta^n\right] \\ &= \mathbb{E}\left[\bar{\theta}^{n-1}\right] - \theta^n. \end{aligned} \tag{6.45}$$

We note that β^{n-1} is the bias computed after iteration $n-1$ (that is, after we have computed $\bar{\theta}^{n-1}$). β^{n-1} is the bias when we use $\bar{\theta}^{n-1}$ as an estimate of θ^n.

The variance of the observation $\hat{\theta}^n$ is computed as follows:

$$\begin{aligned} \mathit{Var}\left[\hat{\theta}^n\right] &= \mathbb{E}\left[\left(\hat{\theta}^n - \theta^n\right)^2\right] \\ &= \mathbb{E}\left[(\varepsilon^n)^2\right] \\ &= \sigma^2. \end{aligned} \tag{6.46}$$

It can be shown (see section 6.7.1) that the optimal stepsize is given by

$$\alpha_{n-1} \quad = \quad 1 - \frac{\sigma^2}{\left(1+\lambda^{n-1}\right)\sigma^2 + (\beta^{n-1})^2}, \tag{6.47}$$

where λ is computed recursively using

$$\lambda^n = \begin{cases} (\alpha_{n-1})^2, & n = 1 \\ (1-\alpha_{n-1})^2\lambda^{n-1} + (\alpha_{n-1})^2, & n > 1. \end{cases} \tag{6.48}$$

The BAKF stepsize formula enjoys several nice properties:

Stationary data For a sequence with a static mean, the optimal stepsizes are given by

$$\alpha_{n-1} \quad = \quad \frac{1}{n} \quad \forall\, n = 1, 2, \ldots\ . \tag{6.49}$$

This is the optimal stepsize for stationary data.

No noise For the case where there is no noise ($\sigma^2 = 0$), we have the following:

$$\alpha_{n-1} \quad = \quad 1 \quad \forall\, n = 1, 2, \ldots. \tag{6.50}$$

This is ideal for nonstationary data with no noise.

Bounded by $1/n$ At all times, the stepsize obeys

$$\alpha_{n-1} \geq \frac{1}{n} \quad \forall\, n = 1, 2, \ldots.$$

This is important since it guarantees asymptotic convergence.

These are particularly nice properties since we typically have to do parameter tuning to get this behavior. The properties are particularly when estimating value functions, since sampled estimates of the value of being in a state tends to be transient.

The problem with using the stepsize formula in equation (6.47) is that it assumes that the variance σ^2 and the bias $(\beta^n)^2$ are known. This can be problematic in real instances, especially the assumption of knowing the bias, since computing this basically requires knowing the real function. If we have this information, we do not need this algorithm.

As an alternative, we can try to estimate these quantities from data. Let

$$\begin{aligned}
(\bar{\sigma}^2)^n &= \text{Estimate of the variance of the error after iteration } n,\\
\bar{\beta}^n &= \text{Estimate of the bias after iteration } n,\\
\bar{\nu}^n &= \text{Estimate of the variance of the bias after iteration } n.
\end{aligned}$$

To make these estimates, we need to smooth new observations with our current best estimate, something that requires the use of a stepsize formula. We could attempt to find an optimal stepsize for this purpose, but it is likely that a reasonably chosen deterministic formula will work fine. One possibility is McClain's formula (equation (6.14)):

$$\eta_n = \frac{\eta_{n-1}}{1+\eta_{n-1}-\bar{\eta}}.$$

A limit point such as $\bar{\eta} \in (0.05, 0.10)$ appears to work well across a broad range of functional behaviors. The property of this stepsize that $\eta_n \rightarrow \bar{\eta}$ can be a strength, but it does mean that the algorithm will not tend to converge in the limit, which requires a stepsize that goes to zero. If this is needed, we suggest a harmonic stepsize rule:

$$\eta_{n-1} = \frac{a}{a+n-1},$$

where a in the range between 5 and 10 seems to work quite well for many dynamic programming applications.

Care needs to be used in the early iterations. For example, if we let $\alpha_0 = 1$, then we do not need an initial estimate for $\bar{\theta}^0$ (a trick we have used throughout). However, since the formulas depend on an estimate of the variance, we still have problems in the second iteration. For this reason, we recommend forcing η_1 to equal 1 (in addition to using $\eta_0 = 1$). We also recommend using $\alpha_n = 1/(n+1)$ for the first few iterations, since the estimates of $(\bar{\sigma}^2)^n, \bar{\beta}^n$ and $\bar{\nu}^n$ are likely to be very unreliable in the very beginning.

Figure 6.8 summarizes the entire algorithm. Note that the estimates have been constructed so that α_n is a function of information available up through iteration n.

Figure 6.9 illustrates the behavior of the bias-adjusted Kalman filter stepsize rule for two signals: very low noise (figure 6.9a) and with higher noise (figure 6.9b). For both cases, the signal starts small and rises toward an upper limit of 1.0 (on average). In both figures, we also show the stepsize $1/n$. For the low-noise case, the stepsize stays quite large. For the high noise case, the stepsize roughly tracks $1/n$ (note that it never goes below $1/n$).

6.4 OPTIMAL STEPSIZES FOR APPROXIMATE VALUE ITERATION

All the stepsize rules that we have presented so far are designed to estimate the mean of a nonstationary series. In this section, we develop a stepsize rule that is specifically designed

Step 0. Initialization:

Step 0a. Set the baseline to its initial value, $\bar{\theta}_0$.

Step 0b. Initialize the parameters - $\bar{\beta}_0, \bar{\nu}_0$ and $\bar{\lambda}_0$.

Step 0c. Set initial stepsizes $\alpha_0 = \eta_0 = 1$, and specify the stepsize rule for η.

Step 0d. Set the iteration counter, $n = 1$.

Step 1. Obtain the new observation, $\hat{\theta}^n$.

Step 2. Smooth the baseline estimate.

$$\bar{\theta}^n = (1 - \alpha_{n-1})\bar{\theta}^{n-1} + \alpha_{n-1}\hat{\theta}^n$$

Step 3. Update the following parameters:

$$\begin{aligned} \varepsilon^n &= \bar{\theta}^{n-1} - \hat{\theta}^n, \\ \bar{\beta}^n &= (1 - \eta_{n-1})\,\bar{\beta}^{n-1} + \eta_{n-1}\varepsilon^n, \\ \bar{\nu}^n &= (1 - \eta_{n-1})\bar{\nu}^{n-1} + \eta_{n-1}(\varepsilon^n)^2, \\ (\bar{\sigma}^2)^n &= \frac{\bar{\nu}^n - (\bar{\beta}^n)^2}{1 + \lambda^{n-1}}. \end{aligned}$$

Step 4. Evaluate the stepsizes for the next iteration.

$$\begin{aligned} \alpha_n &= \begin{cases} 1/(n+1) & n = 1, 2, \\ 1 - \frac{(\bar{\sigma}^2)^n}{\bar{\nu}^n}, & n > 2, \end{cases} \\ \eta_n &= \frac{a}{a + n - 1}. \text{ Note that this gives us } \eta_1 = 1. \end{aligned}$$

Step 5. Compute the coefficient for the variance of the smoothed estimate of the baseline.

$$\bar{\lambda}^n = (1 - \alpha_{n-1})^2\bar{\lambda}^{n-1} + (\alpha_{n-1})^2.$$

Step 6. If $n < N$, then $n = n + 1$ and go to Step 1, else stop.

Figure 6.8 The bias-adjusted Kalman filter stepsize rule.

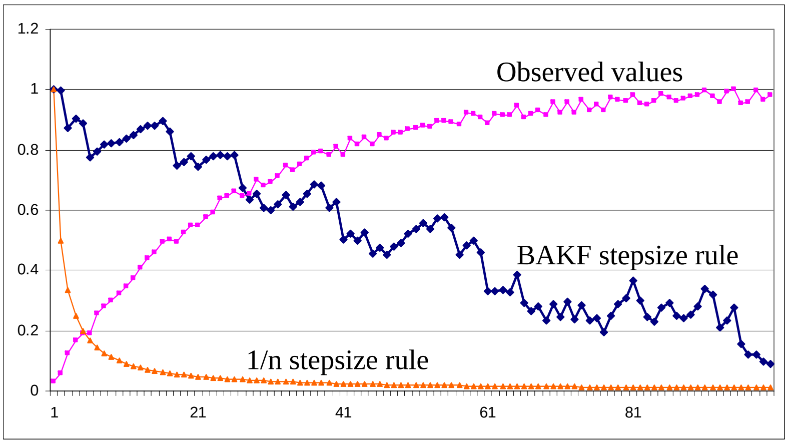
for approximate value iteration, which is an algorithm we are going to see in chapters 17 and 18. We use as our foundation a dynamic program with a single state and single action. We use the same theoretical foundation that we used in section 6.3. However, given the complexity of the derivation, we simply provide the expression.

We start with the basic relationship for our single state problem

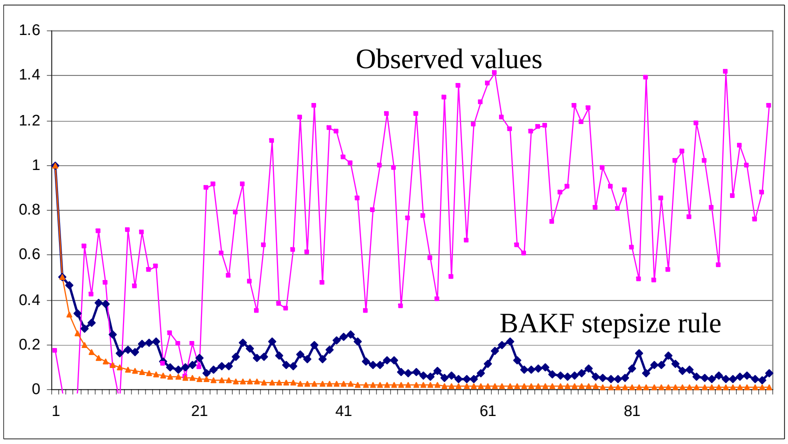
$$v^n(\alpha_{n-1}) = (1 - (1 - \gamma)\alpha_{n-1})v^{n-1} + \alpha_{n-1}\hat{C}^n. \tag{6.51}$$

Let $c = \hat{C}$ be the expected one-period contribution for our problem, and let $Var(\hat{C}) = \sigma^2$. For the moment, we assume c and σ^2 are known. We next define the iterative formulas for two series, λ^n and δ^n, as follows:

$$\begin{aligned} \lambda^n &= \begin{cases} \alpha_0^2 & n = 1 \\ \alpha_{n-1}^2 + (1 - (1 - \gamma)\alpha_{n-1})^2\lambda^{n-1} & n > 1. \end{cases} \\ \delta^n &= \begin{cases} \alpha_0 & n = 1 \\ \alpha_{n-1} + (1 - (1 - \gamma)\alpha_{n-1})\delta^{n-1} & n > 1. \end{cases} \end{aligned}$$



6.9a Bias-adjusted Kalman filter for a signal with low noise.



6.9b Bias-adjusted Kalman filter for a signal with higher noise.

Figure 6.9 The BAKF stepsize rule for low-noise (a) and high-noise (b). Each figure shows the signal, the BAKF stepsizes and the stepsizes produced by the $1/n$ stepsize rule.

It is possible to then show that

$$\begin{aligned} \mathbb{E}(v^n) &= \delta^n c, \\ \mathit{Var}(v^n) &= \lambda^n \sigma^2. \end{aligned}$$

Let $v^n(\alpha_{n-1})$ be defined as in equation (6.51). Our goal is to solve the optimization problem

$$\min_{\alpha_{n-1}} \mathbb{E}\left[\left(v^n(\alpha_{n-1}) - \mathbb{E}\hat{v}^n\right)^2\right] \tag{6.52}$$

The optimal solution can be shown to be given by

$$\alpha_{n-1} = \frac{(1-\gamma)\lambda^{n-1}\sigma^2 + (1-(1-\gamma)\delta^{n-1})^2 c^2}{(1-\gamma)^2\lambda^{n-1}\sigma^2 + (1-(1-\gamma)\delta^{n-1})^2 c^2 + \sigma^2}. \tag{6.53}$$

We refer to equation (6.53) as the *optimal stepsize for approximate value iteration* (OSAVI). Of course, it is only optimal for our single state problem, and it assumes that we know the expected contribution per time period c, and the variance in the contribution $\hat{C}$, σ^2.

OSAVI has some desirable properties. If $\sigma^2 = 0$, then $\alpha_{n-1} = 1$. Also, if $\gamma = 0$, then $\alpha_{n-1} = 1/n$. It is also possible to show that $\alpha_{n-1} \geq (1-\gamma)/n$ for any sample path.

All that remains is adapting the formula to more general dynamic programs with multiple states and where we are searching for optimal policies. We suggest the following adaptation. We propose to estimate a single constant $\bar{c}$ representing the average contribution per period, averaged over all states. If $\hat{C}^n$ is the contribution earned in period n, let

$$\begin{aligned} \bar{c}^n &= (1-\nu_{n-1})\bar{c}^{n-1} + \nu_{n-1}\hat{C}^n, \\ (\bar{\sigma}^n)^2 &= (1-\nu_{n-1})(\bar{\sigma}^{n-1})^2 + \nu_{n-1}(\bar{c}^n - \hat{C}^n)^2. \end{aligned}$$

Here, ν_{n-1} is a separate stepsize rule. Our experimental work suggests that a constant stepsize works well, and that the results are quite robust with respect to the value of ν_{n-1}. We suggest a value of $\nu_{n-1} = 0.2$. Now let $\bar{c}^n$ be our estimate of c, and let $(\bar{\sigma}^n)^2$ be our estimate of σ^2.

We could also consider estimating $\bar{c}^n(s)$ and $(\bar{\sigma}^n)^2(s)$ for each state, so that we can estimate a state-dependent stepsize $\alpha_{n-1}(s)$. There is not enough experimental work to support the value of this strategy, and lacking this we favor simplicity over complexity.

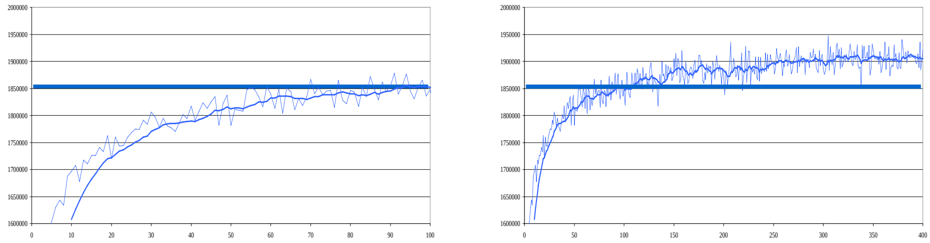
6.5 CONVERGENCE

A practical issue that arises with all stochastic approximation algorithms is that we simply do not have reliable, implementable stopping rules. Proofs of convergence in the limit are an important theoretical property, but they provide no guidelines or guarantees in practice. A good illustration of the issue is given in figure 6.10. Figure 6.10a shows the objective function for a dynamic program over 100 iterations (in this application, a single iteration required approximately 20 minutes of CPU time). The figure shows the objective function for an ADP algorithm which was run 100 iterations, at which point it appeared to be flattening out (evidence of convergence). Figure 6.10b is the objective function for the same algorithm run for 400 iterations. A solid line that shows the best objective function after 100 iterations is shown at the same level on the graph where the algorithm was run for 400 iterations. As we see, the algorithm was nowhere near convergence after 100 iterations.

We refer to this behavior as "apparent convergence," and it is particularly problematic on large-scale problems where run times are long. Typically, the number of iterations needed before the algorithm "converges" requires a level of subjective judgment. When the run times are long, wishful thinking can interfere with this process.

Complicating the analysis of convergence in approximate dynamic programming is the behavior in some problems to go through periods of stability which are simply a precursor to breaking through to new plateaus. During periods of exploration, an ADP algorithm might discover a strategy that opens up new opportunities, moving the performance of the algorithm to an entirely new level.

Special care has to be made in the choice of stepsize rule. In any algorithm using a declining stepsize, it is possible to show a stabilizing objective function simply because the



6.10a: Objective function over 100 iterations.

6.10b: Objective function over 400 iterations.

Figure 6.10 The objective function, plotted over 100 iterations (a), displays "apparent convergence." The same algorithm, continued over 400 iterations (b), shows significant improvement.

stepsize is decreasing. This problem is exacerbated when using algorithms based on value iteration, where updates to the value of being in a state depend on estimates of the values of future states, which can be biased. We recommend that initial testing of an ADP algorithm start with inflated stepsizes. After getting a sense for the number of iterations needed for the algorithm to stabilize, decrease the stepsize (keeping in mind that the number of iterations required to convergence may increase) to find the right tradeoff between noise and rate of convergence.

6.6 GUIDELINES FOR CHOOSING STEPSIZE FORMULAS

Given the plethora of strategies for computing stepsizes, it is perhaps not surprising that there is a need for general guidance when choosing a stepsize formula. Strategies for stepsizes are problem-dependent, and as a result any advice reflects the experience of the individual giving the advice.

We first suggest that the reader review the material in section 17.8. Are you using some variation of approximate value iteration or Q-learning, or are you using a variation of approximate policy iteration for a finite or infinite horizon problem? The implications on stepsizes are significant, and appear to have been largely overlooked in the research community. Stepsizes for variations of policy iteration tend to be closer to $1/n$, while stepsizes for value iteration and Q-learning need to be larger, especially for discount factors closer to 1.

With this in mind, we offer the following general strategies for choosing stepsizes:

Step 1 Start with a constant stepsize α and test out different values. Problems with a relatively high amount of noise will require smaller stepsizes. Periodically stop updating the value function approximation and test your policy. Plot the results to see roughly how many iterations are needed before your results stop improving.

Step 2 Now try the harmonic stepsize $a/(a+n-1)$. $a = 1$ produces the $1/n$ stepsize rule that is provably convergent, but is likely to decline too quickly. $1/n$ absolutely should not be used for approximate value iteration or Q-learning. To choose a, look at how many iterations seemed to be needed when using a constant stepsize. If 100 iterations appears to be enough for a stepsize of 0.1, then try $a \approx 10$, as it produces a stepsize of roughly .1 after 100 iterations. If you need 10,000 iterations, choose $a \approx 1000$. But you will need to tune a. An alternative rule is the polynomial stepsize rule $\alpha = 1/n^{\beta}$ with $\beta \in (0.5, 1]$ (we suggest 0.7 as a good starting point).

Step 3 Now try either the BAKF stepsize rule (section 6.3.3) for policy iteration, LSPE and LSTD, or the OSAVI rule (section 6.4) for approximate value iteration, TD learning and Q-learning. These are both fairly robust, stochastic stepsize rules that adapt to the data. Both have a single tunable parameter (a stepsize), but we have found that the stepsize smoothing for OSAVI is more robust.

There is always the temptation to do something simple. A constant stepsize, or a harmonic rule, are both extremely simple to implement. Keep in mind that both have a tunable parameter, and that the constant stepsize rule will not converge to anything (although the final solution may be quite acceptable). A major issue is that the best tuning of a stepsize not only depends on a problem, but also on the parameters of a problem such as the discount factor. BAKF and OSAVI are more difficult to implement, but are more robust to the setting of the single, tunable parameter. Tunable parameters can be a major headache in the design of algorithms, and it is good strategy to absolutely minimize the number of tunable parameters your algorithm needs. Stepsize rules should be something you code once and forget about.

6.7 WHY DOES IT WORK*

6.7.1 Proof of BAKF stepsize

We now have what we need to derive an optimal stepsize for nonstationary data with a mean that is steadily increasing (or decreasing). We refer to this as the *bias-adjusted Kalman filter* stepsize rule (or BAKF), in recognition of its close relationship to the Kalman filter learning rate. We state the formula in the following theorem:

Theorem 6.7.1. *The optimal stepsizes $(\alpha_m)_{m=0}^n$ that minimize the objective function in equation (6.43) can be computed using the expression*

$$\alpha_{n-1} \quad = \quad 1 - \frac{\sigma^2}{(1+\lambda^{n-1})\,\sigma^2 + (\beta^{n-1})^2}, \tag{6.54}$$

where λ is computed recursively using

$$\lambda^n = \begin{cases} (\alpha_{n-1})^2, & n = 1 \\ (1-\alpha_{n-1})^2\lambda^{n-1} + (\alpha_{n-1})^2, & n > 1. \end{cases} \tag{6.55}$$

Proof: We present the proof of this result because it brings out some properties of the solution that we exploit later when we handle the case where the variance and bias are unknown. Let $F(\alpha_{n-1})$ denote the objective function from the problem stated in (6.43).

$$F(\alpha_{n-1}) \quad = \quad \mathbb{E}\left[\left(\bar{\theta}^n(\alpha_{n-1}) - \theta^n\right)^2\right] \tag{6.56}$$

$$= \quad \mathbb{E}\left[\left((1-\alpha_{n-1})\,\bar{\theta}^{n-1} + \alpha_{n-1}\hat{\theta}^n - \theta^n\right)^2\right] \tag{6.57}$$

$$= \quad \mathbb{E}\left[\left((1-\alpha_{n-1})\left(\bar{\theta}^{n-1} - \theta^n\right) + \alpha_{n-1}\left(\hat{\theta}^n - \theta^n\right)\right)^2\right] \tag{6.58}$$

$$\begin{aligned} = \quad & (1-\alpha_{n-1})^2\,\mathbb{E}\left[\left(\bar{\theta}^{n-1} - \theta^n\right)^2\right] + (\alpha_{n-1})^2\,\mathbb{E}\left[\left(\hat{\theta}^n - \theta^n\right)^2\right] \\ & +2\alpha_{n-1}(1-\alpha_{n-1})\underbrace{\mathbb{E}\left[\left(\bar{\theta}^{n-1} - \theta^n\right)\left(\hat{\theta}^n - \theta^n\right)\right]}_{I}. \end{aligned} \tag{6.59}$$

Equation (6.56) is true by definition, while (6.57) is true by definition of the updating equation for $\bar{\theta}^n$. We obtain (6.58) by adding and subtracting $\alpha_{n-1}\theta^n$. To obtain (6.59), we expand the quadratic term and then use the fact that the stepsize rule, α_{n-1}, is deterministic, which allows us to pull it outside the expectations. Then, the expected value of the cross-product term, *I*, vanishes under the assumption of independence of the observations and the objective function reduces to the following form

$$F(\alpha_{n-1}) = (1-\alpha_{n-1})^2\,\mathbb{E}\left[\left(\bar{\theta}^{n-1}-\theta^n\right)^2\right] + (\alpha_{n-1})^2\,\mathbb{E}\left[\left(\hat{\theta}^n-\theta^n\right)^2\right]. \quad (6.60)$$

In order to find the optimal stepsize, α^*_{n-1}, that minimizes this function, we obtain the first-order optimality condition by setting $\frac{\partial F(\alpha_{n-1})}{\partial \alpha_{n-1}} = 0$, which gives us

$$-2\left(1-\alpha^*_{n-1}\right)\mathbb{E}\left[\left(\bar{\theta}^{n-1}-\theta^n\right)^2\right] + 2\alpha^*_{n-1}\mathbb{E}\left[\left(\hat{\theta}^n-\theta^n\right)^2\right] = 0. \quad (6.61)$$

Solving this for α^*_{n-1} gives us the following result

$$\alpha^*_{n-1} = \frac{\mathbb{E}\left[\left(\bar{\theta}^{n-1}-\theta^n\right)^2\right]}{\mathbb{E}\left[\left(\bar{\theta}^{n-1}-\theta^n\right)^2\right] + \mathbb{E}\left[\left(\hat{\theta}^n-\theta^n\right)^2\right]}. \quad (6.62)$$

Recall that we can write $(\bar{\theta}^{n-1}-\theta^n)^2$ as the sum of the variance plus the bias squared using

$$\mathbb{E}\left[\left(\bar{\theta}^{n-1}-\theta^n\right)^2\right] = \lambda^{n-1}\sigma^2 + \left(\beta^{n-1}\right)^2. \quad (6.63)$$

Using (6.63) and $\mathbb{E}\left[\left(\hat{\theta}^n-\theta^n\right)^2\right] = \sigma^2$ in (6.62) gives us

$$\begin{aligned}\alpha_{n-1} &= \frac{\lambda^{n-1}\sigma^2 + (\beta^{n-1})^2}{\lambda^{n-1}\sigma^2 + (\beta^{n-1})^2 + \sigma^2}\\ &= 1 - \frac{\sigma^2}{\left(1+\lambda^{n-1}\right)\sigma^2 + (\beta^{n-1})^2},\end{aligned}$$

which is our desired result (equation (6.54)). □

From this result, we can next establish several properties through the following corollaries.

Corollary 6.7.1. *For a sequence with a static mean, the optimal stepsizes are given by*

$$\alpha_{n-1} = \frac{1}{n} \quad \forall\, n = 1, 2, \ldots . \quad (6.64)$$

Proof: In this case, the mean $\theta^n = \mu$ is a constant. Therefore, the estimates of the mean are unbiased, which means $\beta^n = 0 \quad \forall t = 2, \ldots,$. This allows us to write the optimal stepsize as

$$\alpha_{n-1} = \frac{\lambda^{n-1}}{1+\lambda^{n-1}}. \quad (6.65)$$

Substituting (6.65) into (6.48) gives us

$$\alpha_n = \frac{\alpha_{n-1}}{1+\alpha_{n-1}}. \tag{6.66}$$

If $\alpha_0 = 1$, it is easy to verify (6.64). □

For the case where there is no noise ($\sigma^2 = 0$), we have the following:

Corollary 6.7.2. *For a sequence with zero noise, the optimal stepsizes are given by*

$$\alpha_{n-1} = 1 \quad \forall\, n = 1, 2, \ldots. \tag{6.67}$$

The corollary is proved by simply setting $\sigma^2 = 0$ in equation (6.47).

As a final result, we obtain

Corollary 6.7.3. *In general,*

$$\alpha_{n-1} \geq \frac{1}{n} \quad \forall\, n = 1, 2, \ldots.$$

Proof: We leave this more interesting proof as an exercise to the reader (see exercise 6.13).

Corollary 6.7.3 is significant since it establishes one of the conditions needed for convergence of a stochastic approximation method, namely that $\sum_{n=1}^{\infty} \alpha_n = \infty$. An open theoretical question, as of this writing, is whether the BAKF stepsize rule also satisfies the requirement that $\sum_{n=1}^{\infty} (\alpha_n)^2 < \infty$.

6.8 BIBLIOGRAPHIC NOTES

Sections 17.8 - 6.2 A number of different communities have studied the problem of "stepsizes," including the business forecasting community (Brown (1959), Holt et al. (1960), Brown (1963), Giffin (1971), Trigg (1964), Gardner (1983)), artificial intelligence (Jaakkola et al. (1994*a*), Darken & Moody (1991), Darken et al. (1992), Sutton & Singh (1994)), stochastic programming (Kesten (1958) , Mirozahmedov & Uryasev (1983) , Pflug (1988), Ruszczyński & Syski (1986)) and signal processing (Goodwin & Sin (1984), Douglas & Mathews (1995)). The neural network community refers to "learning rate schedules"; see Haykin (1999). Even-dar & Mansour (2003) provides a thorough analysis of convergence rates for certain types of stepsize formulas, including $1/n$ and the polynomial learning rate $1/n^{\beta}$, for Q-learning problems. These sections are based on the presentation in Powell & George (2006).

Section 6.3 - This section is based on the review in Powell & George (2006), along with the development of the optimal stepsize rule. Our proof that the optimal stepsize is $1/n$ for stationary data is based on Kmenta (1997).

Section 6.4 - The optimal stepsize for approximate value iteration was derived in Ryzhov et al. (2009).

PROBLEMS

6.1 Use a stochastic gradient algorithm to solve the problem

$$\min_x \frac{1}{2}(X - x)^2,$$

where X is a random variable. Use a harmonic stepsize rule (equation (6.12)) with parameter $\theta = 5$. Perform 1000 iterations assuming that you observe $X^1 = 6, X^2 = 2, X^3 = 5$ (this can be done in a spreadsheet). Use a starting initial value of $x^0 = 10$. What is the best possible formula for θ for this problem?

6.2 Assume we have to order x assets after which we try to satisfy a random demand D for these assets, where D is randomly distributed between 100 and 200. If $x > D$, we have ordered too much and we pay $5(x - D)$. If $x < D$, we have an underage, and we have to pay $20(D - x)$.

(a) Write down the objective function in the form $\min_x \mathbb{E} f(x, D)$.

(b) Derive the stochastic gradient for this function.

(c) Find the optimal solution analytically [Hint: take the expectation of the stochastic gradient, set it equal to zero and solve for the quantity $\mathbb{P}(D \leq x^*)$. From this, find x^*.]

(d) Since the gradient is in units of dollars while x is in units of the quantity of the asset being ordered, we encounter a scaling problem. Choose as a stepsize $\alpha_{n-1} = \alpha_0/n$ where α_0 is a parameter that has to be chosen. Use $x^0 = 100$ as an initial solution. Plot x^n for 1000 iterations for $\alpha_0 = 1, 5, 10, 20$. Which value of α_0 seems to produce the best behavior?

(e) Repeat the algorithm (1000 iterations) 10 times. Let $\omega = (1, \ldots, 10)$ represent the 10 sample paths for the algorithm, and let $x^n(\omega)$ be the solution at iteration n for sample path ω. Let *Var*(x^n) be the variance of the random variable x^n where

$$\overline{V}(x^n) = \frac{1}{10} \sum_{\omega=1}^{10} (x^n(\omega) - x^*)^2$$

Plot the standard deviation as a function of n for $1 \leq n \leq 1000$.

6.3 Show that if we use a stepsize rule $\alpha_{n-1} = 1/n$, then $\bar{\theta}^n$ is a simple average of $\hat{\theta}^1, \hat{\theta}^2, \ldots, \hat{\theta}^n$ (thus proving equation 6.11).

6.4 A customer is required by her phone company to pay for a minimum number of minutes per month for her cell phone. She pays 12 cents per minute of guaranteed minutes, and 30 cents per minute that she goes over her minimum. Let x be the number of minutes she commits to each month, and let M be the random variable representing the number of minutes she uses each month, where M is normally distributed with mean 300 minutes and a standard deviation of 60 minutes.

(a) Write down the objective function in the form $\min_x \mathbb{E} f(x, M)$.

(b) Derive the stochastic gradient for this function.

(c) Let $x^0 = 0$ and choose as a stepsize $\alpha_{n-1} = 10/n$. Use 100 iterations to determine the optimum number of minutes the customer should commit to each month.

6.5 Show that if we use a stepsize rule $\alpha_{n-1} = 1/n$, then $\bar{\theta}^n$ is a simple average of $\hat{\theta}^1, \hat{\theta}^2, \ldots, \hat{\theta}^n$ (thus proving equation 6.11). Use this result to argue that any solution of equation (6.2) produces the mean of W.

6.6 We are going to again try to use approximate dynamic programming to estimate a discounted sum of random variables:

$$F^T = \mathbb{E}\sum_{t=0}^{T}\gamma^t R_t,$$

where R_t is a random variable that is uniformly distributed between 0 and 100 (you can use this information to randomly generate outcomes, but otherwise you cannot use this information). This time we are going to use a discount factor of $\gamma = .95$. We assume that R_t is independent of prior history. We can think of this as a single state Markov decision process with no decisions.

(a) Using the fact that $\mathbb{E}R_t = 50$, give the exact value for F^{100}.

(b) Propose an approximate dynamic programming algorithm to estimate F^T. Give the value function updating equation, using a stepsize $\alpha_t = 1/t$.

(c) Perform 100 iterations of the approximate dynamic programming algorithm to produce an estimate of F^{100}. How does this compare to the true value?

(d) Compare the performance of the following stepsize rules: Kesten's rule, the stochastic gradient adaptive stepsize rule (use $\nu = .001$), $1/n^\beta$ with $\beta = .85$, the Kalman filter rule, and the optimal stepsize rule. For each one, find both the estimate of the sum and the variance of the estimate.

6.7 Consider a random variable given by $R = 10U$ (which would be uniformly distributed between 0 and 10). We wish to use a stochastic gradient algorithm to estimate the mean of R using the iteration $\bar{\theta}^n = \bar{\theta}^{n-1} - \alpha_{n-1}(R^n - \bar{\theta}^{n-1})$, where R^n is a Monte Carlo sample of R in the n^{th} iteration. For each of the stepsize rules below, use equation (5.18) to measure the performance of the stepsize rule to determine which works best, and compute an estimate of the bias and variance at each iteration. If the stepsize rule requires choosing a parameter, justify the choice you make (you may have to perform some test runs).

(a) $\alpha_{n-1} = 1/n$.

(b) Fixed stepsizes of $\alpha_n = .05, .10$ and $.20$.

(c) The stochastic gradient adaptive stepsize rule (equations (6.22)-(6.23)).

(d) The Kalman filter (equations (6.37)-(6.41)).

(e) The optimal stepsize rule (algorithm 6.8).

6.8 Repeat exercise 6.7 using

$$R^n = 10(1 - e^{-0.1n}) + 6(U - 0.5).$$

6.9 Repeat exercise 6.7 using

$$R^n = \left(10/(1 + e^{-0.1(50-n)})\right) + 6(U - 0.5).$$

6.10 Let U be a uniform $[0, 1]$ random variable, and let

$$\mu^n = 1 - \exp(-\theta_1 n).$$

Now let $\hat{R}^n = \mu^n + \theta_2(U^n - .5)$. We wish to try to estimate μ^n using

$$\bar{R}^n = (1 - \alpha_{n-1})\bar{R}^{n-1} + \alpha_{n-1}\hat{R}^n.$$

In the exercises below, estimate the mean (using $\bar{R}^n$) and compute the standard deviation of $\bar{R}^n$ for $n = 1, 2, \ldots, 100$, for each of the following stepsize rules:

- $\alpha_{n-1} = 0.10$.
- $\alpha_{n-1} = \theta/(\theta + n - 1)$ for $a = 1, 10$.
- Kesten's rule.
- The bias-adjusted Kalman filter stepsize rule.

For each of the parameter settings below, compare the rules based on the average error (1) over all 100 iterations and (2) in terms of the standard deviation of $\bar{R}^{100}$.

(a) $\theta_1 = 0, \theta_2 = 10$.

(b) $\theta_1 = 0.05, \theta_2 = 0$.

(c) $\theta_1 = 0.05, \theta_2 = 0.2$.

(d) $\theta_1 = 0.05, \theta_2 = 0.5$.

(e) Now pick the single stepsize that works the best on all four of the above exercises.

6.11 An oil company covers the annual demand for oil using a combination of futures and oil purchased on the spot market. Orders are placed at the end of year $t-1$ for futures that can be exercised to cover demands in year t. If too little oil is purchased this way, the company can cover the remaining demand using the spot market. If too much oil is purchased with futures, then the excess is sold at 70 percent of the spot market price (it is not held to the following year – oil is too valuable and too expensive to store).

To write down the problem, model the exogenous information using

$$\begin{aligned}
\hat{D}_t &= \text{Demand for oil during year } t,\\
\hat{p}^s_t &= \text{Spot price paid for oil purchased in year } t,\\
\hat{p}^f_{t,t+1} &= \text{Futures price paid in year } t \text{ for oil to be used in year } t+1.
\end{aligned}$$

The demand (in millions of barrels) is normally distributed with mean 600 and standard deviation of 50. The decision variables are given by

$$\begin{aligned}
\bar{\theta}^f_{t,t+1} &= \text{Number of futures to be purchased at the end of year } t \text{ to be used in}\\
&\quad\ \text{year } t+1.\\
\bar{\theta}^s_t &= \text{Spot purchases made in year } t.
\end{aligned}$$

(a) Set up the objective function to minimize the expected total amount paid for oil to cover demand in a year $t+1$ as a function of $\bar{\theta}^f_t$. List the variables in your expression that are not known when you have to make a decision at time t.

(b) Give an expression for the stochastic gradient of your objective function. That is, what is the derivative of your function for a particular sample realization of demands and prices (in year $t+1$)?

(c) Generate 100 years of random spot and futures prices as follows:

$$\begin{aligned}\hat{p}_t^f &= 0.80 + 0.10U_t^f,\\ \hat{p}_{t,t+1}^s &= \hat{p}_t^f + 0.20 + 0.10U_t^s,\end{aligned}$$

where U_t^f and U_t^s are random variables uniformly distributed between 0 and 1. Run 100 iterations of a stochastic gradient algorithm to determine the number of futures to be purchased at the end of each year. Use $\bar{\theta}_0^f = 30$ as your initial order quantity, and use as your stepsize $\alpha_t = 20/t$. Compare your solution after 100 years to your solution after 10 years. Do you think you have a good solution after 10 years of iterating?

6.12 The proof in section 5.8.3 was performed assuming that μ is a scalar. Repeat the proof assuming that μ is a vector. You will need to make adjustments such as replacing Assumption 2 with $\|g^n\| < B$. You will also need to use the triangle inequality which states that $\|a+b\| \leq \|a\| + \|b\|$.

6.13 Prove corollary 6.7.3.

CHAPTER 7

DERIVATIVE-FREE STOCHASTIC SEARCH

There are *many* settings where we have to make a choice x to maximize (or minimize) some function that we can represent as $F(x) = \mathbb{E}F(x, W)$ where x is limited to a set of discrete choices such as materials, colors, designs, configurations, and medical treatments. We either do not know (or cannot compute) $\mathbb{E}F(x, W)$, so we depend on sequentially testing the function at discrete points $(x_1, x_2, \ldots, x_M)$. Examples of applications include

- Choosing the best path over a network - After taking a new position and renting a new apartment, you use the internet to identify a set of K paths - many overlapping, but covering modes such as walking, transit, cycling, Uber, and mixtures of these. Each day you get to try a different path x to try to learn the time required μ_x to traverse path x.

- Identifying the best team of basketball players - A coach has 15 players on a basketball team, and has to choose a subset of five for his starting lineup. The players vary in terms of shooting, rebounding and defensive skills.

- Maximizing ad-clicks - A web service has to choose ads to maximize the number of ad-clicks, which reflect the type of ad (automotive? health? travel? food?). The popularity of the ad may reflect the type of product, the way the ad is presented (graphics?), and other information such as time of day.

- Tuning a business simulator - We may have a computer simulation of an Amazon fulfillment center, Uber's dispatch process, or the movement of pilots and aircraft for a charter jet service. All of these applications involve tunable parameters to get the simulator to behave in a way that matches historical performance.

- Repairing/replacing power transformers - Utilities have to manage transformers that range from the small 4-14 kilo-volt transformers you may see on utility poles, to the larger 69kv, and as large as 500 kv, transformers used for transmission networks. Failures can result in catastrophic explosions. The status of a transformer is hard to assess because it is related to both age and the stresses of voltage surges and lightning strikes. An indicator of age is the concentration of gases in the oil that fills transformers. A policy is to repair or replace when the gas level G_t exceeds a threshold θ. The utility can divide θ into ranges, and uses trial and error to find the best threshold.

- Materials science - There are a number of learning problems that are encountered by materials scientists, who have to do considerable trial and error in the laboratory. Some examples are:

  - Find the best catalyst (out of dozens) to maximize the length of a carbon nanotube.
  - Find the best shape and density of nanoparticles to maximize the photoconductivity when the particles are spread across a surface subjected to reflected light.
  - It is possible to store particles in an oil-emulsion bubble that then releases its contents when subjected to X-rays. The problem is to find the best bubble diameter, the best density of nanoparticles on the surface of the bubble, and the X-ray density to optimize the timing of the release of the particles.

- Medical decision making I: choosing drugs - Imagine that we have to find the best diabetes drug for a patient who is not responding well to metformin, the most popular diabetes medication. A diabetes doctor has to choose from among four classes of drug, and then subclasses within each class.

- Medical decision making II: knee replacement options - Knee replacement has become a common surgery, but there are different choices: delay, diet and exercise, rehabilitation without surgery, surgery (and there are different options here), followed by different rehabilitation pathways. Doctors have to find the best treatment options given the characteristics of the patient which may include age, gender, weight, and whether or not the patient smokes.

- Tuning the parameters of an energy storage policy - We have to decide when to store energy from a solar array, when to buy or sell to or from the grid, and how to manage storage to meet the time varying loads of a building. The rules may depend on the price of energy from the grid, the availability of energy from the solar array, and the demand for energy in the building. We may tune these rules in a simulator, but we may be interested in tuning them in the field on an ongoing basis.

Perhaps it is the historical roots of the problems that motivate derivative-based and derivative-free algorithms, but the culture of the derivative-free problem class tends to arise in problems where a function evaluation is relatively expensive. Examples of different costs of function evaluations are:

- An analytical function, such as $\sum_i c_i x_i$, which may take fractions of a second.
- Complex analytical functions, which require large sums or numerical integration, which may take seconds to minutes.
- Computer simulations (of physical or business processes) which take minutes to hours to days (these may take a week or two).
- Laboratory experiments - Testing a new chemical compound or material can take hours to days to a month or more.
- Field experiments - Evaluating the price of a product, testing a new drug or business process, can take days to months, and possibly a year (think of testing a product for the Christmas season, or admissions policies for a university).

In addition, there are settings (especially with field experiments) where it makes sense to evaluate performance based on the cumulative rewards (equation (7.3)) as opposed to just considering the final design (equation (7.2)).

In all of these settings, we have to evaluate the performance of a set of controllable parameters x, which can arise in a number of flavors, including

- Binary - $\mathcal{X} = \{0, 1\}$. Binary choices arise frequently in finance (hold or sell an asset), and internet applications where $x = 0$ means "run the current website" while $x = 1$ means "run the redesigned website" (this is known as A/B testing).
- Discrete - $\mathcal{X} = \{x_1, x_2, \ldots, x_M\}$. This may represent a set of discrete choices (types of materials, sets of features for a product)
- Subset - x may be a vector $(0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1)$ indicating, for example, the starting lineup of a basketball team.
- Discrete vector - x can be a vector of binary elements, or a vector of integers (0, 1, 2, ...).
- Categorical - x may be a category described by a vector of discrete (or possibly discretized) values, which might describe the characteristics of an employee (experience, home location, areas of expertise), the features of a movie, or the attributes of a (potentially threatening) website. If we let $a_1, a_2, \ldots, a_K$ be the different attributes of a choice, we see that the number of possible categories can be extremely large.

At the same time, there are other settings where x is continuous, but where we do not have access to derivatives of either $F(x)$ or even $F(x, W)$ for a sampled value of W (as we used in chapter 5). If x is low-dimensional (generally three or fewer), we might be able to discretize x. If x is higher dimensional, we might simply generate a sample $x_1, \ldots, x_K$ from the (possibly) large set of feasible values of x.

However it is constructed, let $\mathcal{X} = \{x_1, \ldots, x_M\}$ be the set of discrete values of x we can choose from. We can start by formulating the optimization problem using the same asymptotic form we used in chapter 4 which we write

$$\max_{x \in \mathcal{X}} \mathbb{E}\{F(x, W)|S_0\}. \tag{7.1}$$

While this is the standard formulation for gradient-based algorithms, this is not the case for derivative-free settings. Here, the most common formulation is to specify a policy

$X^\pi(S^n)$ that specifies the design $x^n = X^\pi(S^n)$ when our knowledge of the function $\mathbb{E}\{F(x,W)|S_0\}$ is captured by S^n that allows us to create an approximation $\overline{F}^n(x)$. For example, S^n might be a set of estimates $\bar{\mu}_x^n$ for $x \in \mathcal{X}$, along with a measure of the uncertainty in how well $\bar{\mu}_x^n$ approximates $\mathbb{E}\{F(x,W)|S_0\}$.

When our policy specifies $x^n = X^\pi(S^n)$, we then observe $F(x^n, W^{n+1})$, after which we are going to update our belief state to S^{n+1}. The goal now is to find the best policy to solve

$$\max_\pi \mathbb{E}\{F(x^{\pi,N}, W)|S_0\}. \tag{7.2}$$

This, of course, is the final-reward formulation that we discussed in chapter 4. We did not consider a cumulative reward formulation for derivative-based settings, but this is quite common for our derivative-free applications, which we formulate as

$$\max_\pi \mathbb{E}\left\{\sum_{n=0}^{N-1} F(x^{\pi,n}, W^{n+1})|S_0\right\}. \tag{7.3}$$

For example, we might use (7.2) when we are searching for the best solar panel, or a manufacturing process that produces the strongest material. By contrast, we would use (7.3) if we want to find the price that maximizes the revenue from selling a product on the internet, since we have to maximize revenues over time while we are experimenting.

There are some essential differences between the asymptotic formulation in (7.1), the terminal reward formulation in (7.2), and the cumulative reward formulation in (7.3). The asymptotic formulation is searching for a real-valued scalar or vector x which is our final design, or what we might call the implementation decision. In our final-reward formulation (7.2), we are trying to find the best *learning policy* $X^\pi(S)$ to find the best *implementation decision* $x^{\pi,N}$ after N iterations or experiments. The cumulative-reward formulation is naturally online, and as a result we are looking for an *implementation policy* which has to learn while it is implementing.

Regardless of the objective function, we are going to make decisions $x^n = X^\pi(S^n)$ using some policy $X^\pi(S^n)$ that depends on our belief about our function (either (7.2) or (7.3)) that is captured by our state S^n. The belief models, which we introduced in chapter 3, might be lookup tables (with independent or correlated beliefs), parametric (linear or nonlinear), or nonparametric. When our belief state is captured by S^n (e.g. our estimates of parameters), we make a decision to run an experiment with $x^n = X^\pi(S^n)$, and then observe an experimental outcome W^{n+1}, we use the recursive updating formulas from chapter 3 which we represent using

$$S^{n+1} = S^M(S^n, x^n, W^{n+1}).$$

This process produces a sequence of states, actions and information that we will typically write using

$$(S^0, x^0 = X^\pi(S^0), W^1, S^1, x^1 = X^\pi(S^1), W^2, \ldots, S^n, x^n = X^\pi(S^n), W^{n+1}, \ldots).$$

We saw this same sequence for derivative-based stochastic search in chapter 5, but in that setting, S^n was the state of the algorithm, while for the derivative-free problems we consider in this chapter, we depend on creating a belief model about the function, and this is captured in the state S^n.

7.1 OBJECTIVE FUNCTIONS FOR LEARNING POLICIES

We evaluate the performance of our implementation decision x by approximating $\mathbb{E}\{F(x,W)|S^0\}$ in some way. In this section, we address the problem of evaluating the performance of our learning policy $X^\pi(S)$, which depends on the setting in which we are doing our testing. The different objectives for evaluating learning policies include:

Terminal reward - This evaluates the expected value of our function after exhausting our budget of N experiments. As we described in section 5.2, if we capture the modeling uncertainty in S_0, observational uncertainty from the sequence $W^1, \ldots, W^N$ (which is what creates uncertainty in $x^{\pi,N}$), and then finally implementation uncertainty (which we are going to call $\widehat{W}$), we get the expected value of a policy $X^\pi(S)$ as

$$F^\pi \quad = \quad \mathbb{E}_{S_0} E_{W^1,\ldots,W^N|S_0} E_{\widehat{W}|W^1,\ldots,W^N} F(x^{\pi,N}, \widehat{W}).$$

Our objective function, then, is to solve

$$\max_\pi F^\pi. \tag{7.4}$$

Cumulative reward - Here we add up the performance over each experiment. We identify three versions based on our assumptions about the nature of the policy:

Stationary policy - We assume that we have to pick the best single policy to apply over the entire horizon:

$$\max_\pi \mathbb{E} \sum_{n=0}^{N-1} F(X^\pi(S^n), W^{n+1}). \tag{7.5}$$

where $S^{n+1} = S^M(S^n, x^n = X^\pi(S^n), W^{n+1})$. Remember that S^n is the belief state after the nth experiment, which means that the transition function $S^M(S^n, x^n, W^{n+1})$ refers to either our frequentist or Bayesian updating equations.

Nonstationary policy - In this setting, the policy depends on time (the iteration counter), but still has to be chosen before we start any of the experiments:

$$\max_{\pi^0,\ldots,\pi^{N-1}} \mathbb{E} \sum_{n=0}^{N-1} F(X^{\pi^n}(S^n), W^{n+1}). \tag{7.6}$$

Below we illustrate a nonstationary policy that depends on $N-n$, which puts more emphasis on exploring different options when $N-n$ is large, while focusing more on choices that appear to be best as we get close to the end of the horizon.

Adaptive policy - Assume we have some form of parametric policy $X^\pi(S^n|\theta^n)$ parameterized by θ^n. Further assume that we update θ^n as new information comes in using the policy Θ^π where $\theta^{n+1} = \Theta^\pi(S^n, \theta^n)$ (technically θ^n would be part of the state variable, but we write it explicitly for clarity).

$$\max_{\pi^\theta} \max_\pi \mathbb{E} \sum_{n=0}^{N-1} F(X^\pi(S^n|\theta^n), W_{n+1}).$$

where $\theta^n = \Theta^{\pi^\theta}(S^n)$ and $S^{n+1} = S^M(S^n, x^n = X^\pi(S^n|\theta^n), W^{n+1})$. Now we have to find the best policy $\Theta^{\pi^\theta}(S^n, \theta^n)$ for updating θ, as well as the best policy $X^\pi(S^n|\theta^n)$ for making a decision (we assume both of these policies are stationary and are chosen in advance).

If we are using the terminal reward form of the objective function, our problem becomes a classical problem known as *ranking and selection*, where the goal is to find the best choice or design, $x^{\pi,N}$, within the budget of N iterations. The ranking and selection is generally encountered in what are known as *offline* learning settings, which typically refers to problems where we do not have to experience our decisions while we are searching.

The cumulative reward formulation in equation (7.5) arises in what are generally known as *online* learning settings, because we have to experience the results of our decisions as they happen. This objective is the classic setting of what is known as the *multiarmed bandit problem.*

7.2 LOOKUP TABLE BELIEF MODELS

We assume throughout that $x \in \mathcal{X} = \{x_1, \ldots, x_M\}$ is one of a set of discrete choices, although it can be a very large set. As before, we let

$$\mu_x = \mathbb{E}\{F(x, W)|S^0\}$$

be the true value of our function given an implementation decision x. Our goal is to find x that solves one of the objective functions presented in section 7.1. Policies depend on our belief about μ_x for $x \in \mathcal{X}$, which we represent as $\bar{\mu}^n_x$ after running n observations of the function $F(x, W)$.

We begin our presentation of derivative-free learning using a lookup table belief model, which offers the simplest setting for illustrating different types of updating strategies. We can use either frequentist or Bayesian belief models, summarized below. For readers with a reasonable understanding of the approximation methods presented in chapter 3, this section can be skipped.

7.2.1 Frequentist belief model

If we use a frequentist belief model, then we do not know anything about μ_x until we perform an initial set of experiments. Assume that out of a total budget of n experiments, that we sample alternative x N^n_x times, observing $\hat{F}^n_x = F(x, W^n)$ each time. We would obtain our initial estimates from

$$\bar{\mu}^n_x = \frac{1}{N^n_x} \sum_{m=1}^{N^n_x} \hat{F}^m_x \tag{7.7}$$

$$\hat{\sigma}^{2,n}_x = \frac{1}{N^n_x - 1} \sum_{m=1}^{N^n_x} (\hat{F}^m_x - \bar{\mu}^n_x)^2. \tag{7.8}$$

As n grows, $\hat{\sigma}^{2,n}_x$ converges to the true variance of $\hat{F}_x$. Sometimes we need the variance of the estimate $\bar{\mu}^n_x$ which would be given by

$$\bar{\sigma}^{2,n}_x = \frac{1}{N^n_x} \hat{\sigma}^{2,n}_x.$$

Note that $\bar{\sigma}^{2,n}_x \to 0$ as $N^n_x \to \infty$.

These estimates can be updated recursively. To simplify notation, fix x (so all estimates are a function of x) and let n be the number of times we have observed x (which we called N^n_x above). If we have $\bar{\mu}^{n-1}$ and $\hat{\sigma}^{2,n-1}$, and then observe $\hat{F}^n$, we would obtain

$$\bar{\mu}^n = \left(1 - \frac{1}{n}\right)\bar{\mu}^{n-1} + \frac{1}{n}\hat{F}^n, \tag{7.9}$$

$$\hat{\sigma}^{2,n} = \begin{cases} \frac{1}{n}(W^n - \bar{\mu}^{n-1})^2 & n = 2, \\ \frac{n-2}{n-1}\hat{\sigma}^{2,n-1} + \frac{1}{n}(W^n - \bar{\mu}^{n-1})^2 & n > 2. \end{cases} \tag{7.10}$$

These statistics can be used to build confidence intervals around μ_x.

7.2.2 Bayesian belief model

If we use a Bayesian belief model, then we assume that we are given a prior, which we might assume is normally distributed, where

$$\mu_x \sim N(\bar{\mu}^0_x, \beta^0_x),$$

where β^0_x is the initial precision, given by $\beta^0_x = 1/\sigma^{2,0}_x$. For example, $\bar{\mu}^0_x$ might be an initial best guess at the sales volume for selling a product at price x, based on our experience selling similar products. The standard deviation σ^0_x captures the spread in what we think the true sales might be, again based on our experience with selling comparable products.

Next assume we run an experiment x and observe

$$\hat{F}^n_x = f(x) + \varepsilon^n_x,$$

where $f(x) = \mathbb{E}\{F(x,W)|S^0\}$, and where $\varepsilon_x \sim N(0, (\sigma^W_x)^2)$. We are going to find it convenient to use the precision of the noise

$$\beta^W_x = \frac{1}{(\sigma^W_x)^2}.$$

Our notation allows the variance to depend on x. When this is the case, we say that the model is *heteroscedastic*, because the variability depends on x. Sometimes we can assume that the variance of noise in an observation of $f(x)$ is independent of x, in which case we would write the variance as $(\sigma^W)^2$. When this is the case, we say that the model is *homoscedastic*.

Assume for the moment that our estimates of μ_x and $\mu_{x'}$ are independent. If we choose $x = x^n$, where we observe $\hat{F}^{n+1}_x$, we can update our estimates using

$$\bar{\mu}^{n+1}_x = \frac{\beta^n_x \bar{\mu}^n_x + \beta^W_x \hat{F}^{n+1}_x}{\beta^n_x + \beta^W_x}, \tag{7.11}$$

$$\beta^{n+1}_x = \beta^n_x + \beta^W_x. \tag{7.12}$$

There are many applications (including virtually all of the examples described in the introduction to this chapter) where the beliefs about two alternatives are not independent. Following our earlier presentation in section 3.4.2, let Σ^0 be our prior covariance, with element

$$\Sigma^n_{xy} = Cov^n(\mu_x, \mu_y).$$

We then define the *precision matrix*

$$B^0 = (\Sigma^0)^{-1}.$$

which plays the same role as the precision β^0_x.

Let e_x be a column vector of zeroes with a 1 for element x, and as before we let $\hat{F}^{n+1}_x$ be the observation when we decide to evaluate alternative x. Our updating equation for the vector $\bar{\mu}^n$ is

$$\bar{\mu}^{n+1}_x = (B^{n+1})^{-1}\left(B^n\bar{\mu}^n_x + \beta^W\hat{F}^{n+1}_x e_{x^n}\right),$$

where B^{n+1} is given by

$$B^{n+1} = (B^n + \beta^W_x e_{x^n}(e_{x^n})^T).$$

Note that $e_x(e_x)^T$ is a matrix of zeroes with a one in row x, column x, whereas β^W is a scalar giving the precision of our measurement W.

Learning with correlated beliefs is a particularly powerful strategy in practical applications, largely because most real applications (especially those with many alternatives) are going to exhibit correlations. Of course, a separate issue is how the correlations are handled when we design a learning policy.

7.2.3 Frequentist or Bayesian?

Statisticians have a long history of being divided into two camps known as "frequentists" and "Bayesians." While it is true that these represent two very different perspectives of uncertainty, in practice the best view reflects the nature of the problem you are working on. In a nutshell, frequentist statistics is most useful when observations are relatively easy to collect and there is not a source of information for creating a prior. Bayesian statistics, on the other hand, is most natural when observations are difficult, and where there is a natural source of information about the parameters before any data has been collected.

Examples where frequentist statistics is probably the best choice are:

■ **EXAMPLE 7.1**

An internet company needs to design a policy for guiding the choice of which ads to display to maximize ad-clicks. There are many ads, without any source of information for deciding which ads are best (in advance). At the same time, it is relatively easy to test a new ad to get an estimate of its popularity.

■ **EXAMPLE 7.2**

A large retail chain needs to identify the dress styles that are selling the best. With hundreds of stores, it is relatively easy for the chain to test market different styles and colors in selected stores.

Examples where Bayesian statistics is more natural include:

■ **EXAMPLE 7.1**

A laboratory scientist needs to find the best combination of temperatures, concentrations and catalysts to maximize the production of a type of nanotube. It takes a day to run a single experiment. The scientist is guided by an understanding of the chemistry of the problem, which she draws on to guide early experiments.

■ **EXAMPLE 7.2**

An internet retailer wants to determine the best price for a new (and expensive) technical textbook. It is necessary to observe sales of the book for several weeks to obtain a sense of how the market is responding to price. Fortunately, the retailer is guided by past experience with the sales of other textbooks.

There are, of course, instances where information is expensive, but where there is no natural source of information to form a prior. In such cases, there is no alternative to running a handful of random experiments. At this point, it is possible to progress using a purely frequentist belief model. However, some will simply prefer the characteristics of a Bayesian belief model, and use these initial experiments to form a prior. This strategy is known as *empirical Bayes*.

7.3 DESIGNING POLICIES

There are two fundamental strategies for finding policies (which may even be optimal, or enjoy strong theoretical guarantees):

Policy search - Here we search over a parameterized class of policies to find the policy (within a class) that optimizes some objective (and here we can use any of the objective functions discussed above). We divide these into two broad classes:

Policy function approximations (PFAs) - These are analytical functions that map states to actions.

Cost function approximations (CFAs) - Here we optimize some parametrically modified cost function to find the best decision.

Policies based on lookahead approximations - These are policies that are based on approximating the impact of a decision now on the future. Lookahead policies can be divided into two broad classes:

Policies based on value function approximations (VFAs) - A value function approximation is a statistically estimated function that captures the value of moving to a downstream state. The value function estimates the contribution of decisions made after landing in a state.

Direct lookahead policies (DLAs) - Here we optimize over an entire trajectory in the future before making a decision now.

Policies based on policy search tend to be simpler in structure and therefore easier to compute, but require tuning. Policies based on lookahead approximations tend to be harder to compute, but in their most basic form do not have any tunable parameters. Of course, it is often possible to introduce a tunable parameter in a lookahead policy to overcome some limitation of the lookahead policy, but at a price of now having to tune the policy.

These four classes of policies cover every policy that has been proposed in the literature for stochastic optimization problems. For example, when we make the transition to dynamic programs for state-dependent functions (starting in chapter 8), we organize all the methods for solving this rich problem class along the lines of these four classes. The presentation here, on a much simpler problem class, provides a nice introduction to this organization.

We now undertake a summary of some of the most popular policies using for derivative-free stochastic optimization, organized along these four classes.

7.3.1 Policy function approximations

A PFA is any function that maps directly from a state to an action. PFAs are popular as belief models for approximating functions we are optimizing, but are less common as policies for solving a derivative-free stochastic search problem. These are more likely to be used in the early stages learning about a function. PFAs, which may be lookup tables or parametric functions, are fixed mappings that determine the decision (or design) to choose next, without resorting to any sort of search (which would be classified as a CFA).

Examples of PFAs are:

Lookup tables Assume that the system can be described by a set of discrete states $\mathcal{S} = \{s_1, s_2, \ldots, s_I\}$, where a state s_i could be:

- The state s_i describes the gender, age and diagnosis of a patient. The policy $X^\pi(S_t)$ might specify the dosage for a medical condition.
- If s_i describes a chess board, an expert chess player knows what to play next.

Parametric functions There are many applications in engineering where a policy (such as the force applied to a robot arm) can be written as a linear function of the state (which captures location and velocity), which we might write as

$$U^\pi(S_t|\theta) = \theta_0 + \theta_1\phi_1(S_t) + \theta_2\phi_2(S_t). \tag{7.13}$$

This is known as an “affine policy” or more descriptively, a “linear decision rule” (or LDR). We might also use a policy with this structure to control the release of water from a reservoir. An example of a nonlinear model is a popular policy for ordering inventory, where we trigger an order if the inventory S_t falls below a minimum θ^L, at which point we order an amount to bring the inventory up to θ^U:

$$X^\pi(S_t|\theta) = \begin{cases} \theta^U - S_t & \text{If } S_t < \theta^L, \\ 0 & \text{Otherwise.} \end{cases}$$

Other examples including selling a stock when its price falls below its 30-day moving average by θ^{sell}, deciding to wear a jacket when the temperature falls below some level θ^{temp}, or to add θ^{slack} to the estimated travel time to ensure that you arrive on time. Finally, a popular approach to solve engineering control problems is to use a neural network to map a multidimensional state variable to a multidimensional control.

Locally parametric functions Computer scientists have designed effective robotic controllers by defining linear policies such as that given in equation (7.13) over local regions which are often designed by hand.

7.3.2 Cost function approximations

Cost function approximations describe policies where we have to minimize something to find the alternative to try next, and where we do not make any effort at approximating the impact of a decision now on the future. Not surprisingly, CFAs cover a wide range of practical, and surprisingly powerful, policies.

Simple greedy policies - We use the term “simple greedy policy” to refer to a policy which chooses an action which maximizes the expected reward given current beliefs, which would be given by

$$X^{SG}(S^n) = \arg\max_x \bar{\mu}_x^n.$$

Now imagine that we have a nonlinear function $F(x, \theta)$ where θ is an unknown parameter where, after n experiments, might be normally distributed with distribution $N(\theta^n, \sigma^{2,n})$. Our simple greedy policy would solve

$$\begin{aligned} X^{SG}(S^n) &= \arg\max_x F(x, \theta^n), \\ &= \arg\max_x F(x, \mathbb{E}(\theta|S^n)). \end{aligned}$$

This describes a classical approach known under the umbrella as *response surface methods* where we pick the best action based on our latest statistical approximation of a function.

Bayes greedy - Bayes greedy is just a greedy policy where the expectation is kept on the outside of the function (where it belongs), which would be written

$$X^{BG}(S^n) = \arg\max_x \mathbb{E}\{F(x, \theta)|S^n\}.$$

When the function $F(x, \theta)$ is nonlinear in θ, this expectation can be tricky to compute.

Upper confidence bounding - UCB policies, which are very popular in computer science, come in many flavors, but they all share a form that follows one of the earliest UCB policies given by

$$\nu_x^{UCB,n} = \bar{\mu}_x^n + 4\sigma^W \sqrt{\frac{\log n}{N_x^n}}, \tag{7.14}$$

where $\bar{\mu}_x^n$ is our estimate of the value of alternative x, and N_x^n is the number of times we evaluate alternative x within the first n iterations. The coefficient $4\sigma^W$ has a theoretical basis, but is typically replaced with a tunable parameter θ^{UCB} which we might write as

$$\nu_x^{UCB,n}(\theta^{UCB}) = \bar{\mu}_x^n + \theta^{UCB} \sqrt{\frac{\log n}{N_x^n}}. \tag{7.15}$$

The UCB policy, then, would be

$$X^{UCB}(S^n|\theta^{UCB}) = \arg\max_x \nu_x^{UCB,n}(\theta^{UCB}). \tag{7.16}$$

UCB policies all use an index comprised of a current estimate of the value of alternative (“arm” in the language of the bandit-oriented UCB community), given by $\bar{\mu}^n_x$, plus a term that encourages exploration. As the number of observations grows, $\log n$ also grows (but logarithmically), while N^n_x counts how many times we have sampled alternative x.

Interval estimation Interval estimation sets the value of a decision to the 90^{th} or 95^{th} percentile of the estimate of the value of a decision. The interval estimation policy is then given by

$$X^{IE}(S^n|\theta^{IE}) = \arg\max_x \left(\bar{\mu}^n_x + \theta^{IE}\bar{\sigma}^n_x\right). \tag{7.17}$$

Here, $\bar{\sigma}^n_x$ is our estimate of the standard deviation of $\bar{\mu}^n_x$. As the number of times we observe action a goes to infinity, $\bar{\sigma}^n_x$ goes to zero. The parameter θ^{IE} is a tunable parameter, although it is common to choose values around 2 or 3. This has the effect of valuing each action at its 90^{th} or 95^{th} percentile.

Boltzman exploration A different form of maximizing over actions involves computing a probability that we pick an action x, given an estimate $\bar{\mu}^n_x$ of the reward from this action. This is typically computed using

$$p^n(x|\theta) = \frac{e^{\theta\bar{\mu}^n_x}}{\sum_{x'} e^{\theta\bar{\mu}^n_{x'}}}. \tag{7.18}$$

Now pick x^n at random according to the distribution $p^n(x|\theta)$. Boltzmann exploration is sometimes referred to as “soft max” since it is performing a maximization in a probabilistic sense.

Both PFAs and CFAs require tuning of a parameter vector θ, where we have to use one of our objective functions (7.4), (7.5), (7.6) or (7.7). We remind the reader that these policies are being used to find the best x to optimize $\mathbb{E}F(x, W)$ in either the terminal reward or cumulative reward forms. However, finding the best policy to maximize this function is itself an optimization problem. So we need a policy to find the best *learning policy* to find the best implementation decision.

7.3.3 Policies based on value function approximations

We first saw value functions in the setting of discrete Markov decision problems (see equation (2.14) in chapter 2). Value functions capture the value of being in a state, which creates a compact way of looking into the future. Value functions can be very powerful for problems where there is a physical state, as might happen when we are optimizing a path over a graph or allocating resources. In the setting of pure learning problems, value functions have not proven to be as useful, but this does not mean that they cannot be used.

Although we are not going to deal with Bellman’s equation in depth until chapter 14, the basic idea (as we saw in equation (2.14)), is quite simple. If $V^{n+1}(S^{n+1})$ is the value of being in state S^{n+1} after $n+1$ observations of our function, then we can compute $V^n(S^n)$ using

$$V^n(S^n) \quad = \quad \max_{x\in\mathcal{X}} \left(C(S^n, x) + E\{V^{n+1}(S^{n+1})|S^n, x\}\right). \tag{7.19}$$

This equation is easy to visualize (and compute) if the state S^n is a node in a graph, or the number of discrete items we are holding in inventory. For learning problems, however, S^n is our belief state, which means the set of probability distributions about all the alternatives. Assume for the moment that our beliefs about each μ_x is normally distributed with mean $\bar{\mu}^n_x$ and variance $s^{2,n}_x$. Our belief state then, would be

$$S^n = (\bar{\mu}^n_x, \sigma^{2,n}_x)_{x \in \mathcal{X}}.$$

Keeping in mind that we have to implicitly assume a distribution (such as normal), our belief state is exactly what is illustrated in figure 7.3.

As a rule, $\bar{\mu}^n_x$ and $\sigma^{2,n}_x$ are continuous, and S^n is multidimensional, since there are estimates for each $x \in \mathcal{X} = \{1, \ldots, M\}$. Not surprisingly, computing Bellman's equation from (7.19) is completely intractable for learning problems.

Below (section 7.6) we are going to introduce an approach known as *Gittins indices* which was an early breakthrough in learning that is based on the use of value functions. Gittins indices can, for certain problems, produce optimal policies (a rarity in this field), but they are difficult to compute and simpler policies have been found to be more useful for problems that arise in practice. There are, however, special problems where value functions, which guide decisions by capturing the downstream value of a learning decision now, do add value.

Lookahead policies in general, however, have proven to be useful. We next discuss the important class of single-period lookahead policies, which have been found to work quite well for some problems, followed by a brief discussion of multi-period lookahead policies.

7.3.4 Single period lookahead policies

A single period lookahead would never work well if we had to deal with a physical state (imagine solving a shortest path problem over a graph with a single period lookahead). However, they often work exceptionally well in the setting of learning problems.

Knowledge gradient The most common form of single-period lookahead policy are value of information policies, which maximize the value of information from a single experiment. Let S^n be our belief state now, and let $S^{n+1}(x)$ be the random belief state if we run an experiment with design x, but before we have observed the outcome. If we are trying to maximize a nonlinear function $F(x, \theta)$, after n experiments we would write

$$E\{F(x, \theta)|S^n\} = F(x, \theta^n),$$

where θ^n is our current estimate of θ (this is captured by our state S^n). If we were to stop now, we would solve

$$\max_{x'} F(x', \theta^n).$$

Now imagine running experiment x, where we will make a noisy observation of $F(x, \theta)$ (note that θ is our uncertain, unknown true parameter). This will produce an updated estimate $\theta^{n+1}(x)$ which is random and depends on the experiment x that we are thinking of running. The quality of our solution after this experiment (given what we know at time n) is given by

$$\mathbb{E}\{\max_{x'} F(x', \theta^{n+1}(x))|S^n\}.$$

We can expect that our experiment using setting x would improve our solution, so we can evaluate this improvement using

$$\nu^{KG}(x) = \mathbb{E}\{\max_{x'} F(x', \theta^{n+1}(x))|S^n\} - \max_{x'} F(x', \theta^n).$$

The quantity $\nu^{KG}(x)$ is sometimes referred to as the *knowledge gradient*, and it gives the expected value of the information from experiment x. This calculation is made by looking one experiment into the future. We cover knowledge gradient policies in considerably greater depth below.

Expected improvement - Known as EI in the literature, expected improvement is a close relative of the knowledge gradient, given by the formula

$$\nu_x^{EI,n} = \mathbb{E}\left[\max\left\{0, \mu_x - \max_{x'} \bar{\mu}^n_{x'}\right\}\middle| S^n, x = x^n\right]. \tag{7.20}$$

Unlike the knowledge gradient, EI does not explicitly capture the value of an experiment, which requires evaluating the ability of an experiment to change the final design decision. Rather, it measures the degree to which an alternative x *might* be better. It does this by capturing the degree to which the random truth μ_x *might* be greater than the current best estimate $\max_{x'} \bar{\mu}^n_{x'}$.

Sequential kriging - This is a methodology developed in the geosciences to guide the investigation of geological conditions, which are inherently continuous and two-dimensional. Kriging evolved in the setting of geo-spatial problems where x is continuous (representing a spatial location, or even a location underground in three dimensions). For this reason, we let the truth be the function $\mu(x)$, rather than μ_x (the notation we used when x was discrete).

Kriging uses a form of meta-modeling where the surface is assumed to be represented by a linear model, a bias model and a noise term which can be written as

$$\mu(x) = \sum_{f\in\mathcal{F}} \theta_f \phi_f(x) + Z(x) + \varepsilon,$$

where $Z(x)$ is the bias function and $(\phi_f(x))_{f\in\mathcal{F}}$ are a set of features extracted from data associated with x. Given the (assumed) continuity of the surface, it is natural to assume that $Z(x)$ and $Z(x')$ are correlated with covariance

$$Cov(Z(x), Z(x')) = \beta \exp\left[-\sum_{i=1}^{d} \alpha_i (x_i - x_i')^2\right],$$

where β is the variance of $Z(x)$ while the parameters α_i perform scaling for each dimension.

The best linear model, which we denote $\bar{Y}^n(x)$, of our surface $\mu(x)$, is given by

$$\begin{aligned}\bar{Y}^n(x) &= \sum_{f\in\mathcal{F}} \theta_f^n \phi_f(x) + \\ &\quad \sum_{i=1}^{n} Cov(Z(x_i), Z(x)) \sum_{j=1}^{n} Cov(Z(x_j), Z(x))(\hat{y}_i - \sum_{f\in\mathcal{F}} \theta_f^n \phi_f(x)),\end{aligned}$$

where θ^n is the least squares estimator of the regression parameters, given the n observations $\hat{y}^1, \ldots, \hat{y}^n$.

Kriging starts with the expected improvement in equation (7.20), with a heuristic modification to handle the uncertainty in an experiment (ignored in (7.20)). This gives an adjusted EI of

$$\mathbb{E}^n I(x) = \mathbb{E}^n \left[\max(\bar{Y}^n(x^{**}) - \mu(x), 0)\right] \left(1 - \frac{\sigma_\varepsilon}{\sqrt{\sigma^{2,n}(x) + \sigma_\varepsilon^2}}\right), \qquad (7.21)$$

where x^{**} is a point chosen to maximize a utility that might be given by

$$u^n(x) = -(\bar{Y}^n(x) + \sigma^n(x)).$$

Since x is continuous, maximizing $u^n(x)$ over x can be hard, so we typically limit our search to previously observed points

$$x^{**} = \arg \max_{x \in \{x^1, \ldots, x^n\}} u^n(x).$$

The expectation in (7.21) can be calculated analytically using

$$\begin{aligned}\mathbb{E}^n \left[\max(\bar{Y}^n(x^{**}) - \mu(x), 0)\right] \quad &= \quad (\bar{Y}^n(x^{**}) - \bar{Y}^n(x))\Phi\left(\frac{\bar{Y}^n(x^{**}) - \bar{Y}^n(x)}{\sigma^n(x)}\right) \\ &\quad + \sigma^n(x)\phi\left(\frac{\bar{Y}^n(x^{**}) - \bar{Y}^n(x)}{\sigma^n(x)}\right),\end{aligned}$$

where $\phi(z)$ is the standard normal density, and $\Phi(z)$ is the cumulative density function for the normal distribution.

Thompson sampling - Thompson sampling works by sampling from the current belief about $\mu_x \sim N(\bar{\mu}_x^n, \bar{\sigma}_x^{n,2})$, which can be viewed as the prior distribution for experiment $n+1$ (some refer to this as the posterior distribution, given the observations $W^1, \ldots, W^n$). Now choose a sample $\hat{\mu}_x^n$ from the distribution $N(\bar{\mu}_x^n, \bar{\sigma}_x^{n,2})$. We view $\hat{\mu}_x^n$ as a simulation of what might happen in the next sample, hence its classification as a one-period lookahead policy. The Thompson sampling policy is then given by

$$X^{TS}(S^n) = \arg\max_x \hat{\mu}_x^n.$$

Thompson sampling is more likely to choose the alternative x with the largest $\bar{\mu}_x^n$, but because we sample from the distribution, we may also choose other alternatives, but are unlikely to choose alternatives where the estimate $\bar{\mu}_x^n$ is low relative to the others.

Value of information policies are well-suited to problems where information is expensive, since they focus on running the experiments with the highest value of information. These policies are particularly effective when the value of information is concave, which means that the marginal value of each additional experiment is lower than the previous one. This property is not always true, especially when experiments are noisy. We revisit this issue below in section 7.7.3.

Player	No. hits	No. at-bats	Average
A	36	100	0.360
B	1	3	0.333
C	7	22	0.318

Table 7.1 History of hitting performance for three candidates.

7.3.5 Multiperiod lookahead policies

While there are problems where a one-period lookahead works quite well, this is not always the case. One problem class where one-period lookaheads perform poorly is when the value of information of a single experiment is low. This happens when the noise from a single experiment is high enough that a single experiment does not help improve decisions. A second problem class is when you are learning, but also have to deal with a physical state.

We illustrate a multiperiod lookahead policy for learning using the setting of trying to identify the best hitter on a baseball team. The only way to collect information is to put the hitter into the lineup and observe what happens. We have an estimate of the probability that the player will get a hit, but we are going to update this estimate as we make observations (this is the essence of learning).

Assume that we have three candidates for the position. The information we have on each hitter from previous games is given in Table 7.1. If we choose player A, we have to balance the likelihood of getting a hit, and the value of the information we gain about his true hitting ability, since we will use the event of whether or not he gets a hit to update our assessment of his probability of getting a hit. We are going to again use Bayes' theorem to update our belief about the probability of getting a hit. Fortunately, this model produces some very intuitive updating equations. Let H^n be the number of hits a player has made in n at-bats. Let $\hat{H}^{n+1} = 1$ if a hitter gets a hit in his $(n+1)$st at-bat. Our prior probability of getting a hit after n at-bats is

$$\mathbb{P}[\hat{H}^{n+1} = 1 | H^n, n] = \frac{H^n}{n}.$$

Once we observe $\hat{H}^{n+1}$, it is possible to show that the posterior probability is

$$\mathbb{P}[\hat{H}^{n+2} = 1 | H^n, n, \hat{H}^{n+1}] = \frac{H^n + \hat{H}^{n+1}}{n+1}.$$

In other words, all we are doing is computing the batting average (hits over at-bats).

Our challenge is to determine whether we should try player A, B or C right now. At the moment, A has the best batting average of 0.360, based on a history of 36 hits out of 100 at-bats. Why would we try player B, whose average is only 0.333? We easily see that this statistic is based on only three at-bats, which would suggest that we have a lot of uncertainty in this average.

We can study this formally by setting up the decision tree shown in Figure 7.1. For practical reasons, we can only study a problem that spans two at-bats. We show the current prior probability of a hit, or no hit, in the first at-bat. For the second at-bat, we show only the probability of getting a hit, to keep the figure from becoming too cluttered.

Figure 7.2 shows the calculations as we roll back the tree. Figure 7.2(c) shows the expected value of playing each hitter for exactly one more at-bat using the information

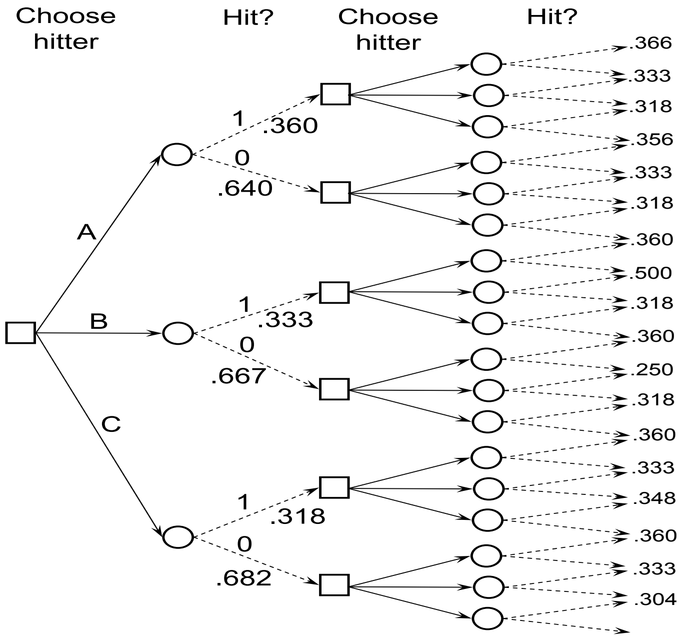


Figure 7.1 The decision tree for finding the best hitter.

obtained from our first decision. It is important to emphasize that after the first decision, only one hitter has had an at-bat, so the batting averages only change for that hitter. Figure 7.2(b) reflects our ability to choose what we think is the best hitter, and Figure 7.2(a) shows the expected value of each hitter before any at-bats have occurred. We use as our reward function the expected number of total hits over the two at-bats. Let R_x be our reward if batter x is allowed to hit, and let H_{1x} and H_{2x} be the number of hits that batter x gets over his two at-bats. Then

$$R_x = H_{1x} + H_{2x}.$$

Taking expectations gives us

$$\mathbb{E}R_x = \mathbb{E}H_{1x} + \mathbb{E}H_{2x}$$

So, if we choose batter A, the expected number of hits is

$$\begin{aligned} \mathbb{E}R_A &= .360(1 + .366) + .640(0 + .356) \\ &= .720 \end{aligned}$$

where 0.360 is our prior belief about his probability of getting a hit; .366 is the expected number of hits in his second at-bat (the same as the probability of getting a hit) given that he got a hit in his first at-bat. If player A did not get a hit in his first at-bat, his updated probability of getting a hit, 0.356, is still higher than any other player. This means that if we have only one more at-bat, we would still pick player A even if he did not get a hit in his first at-bat.

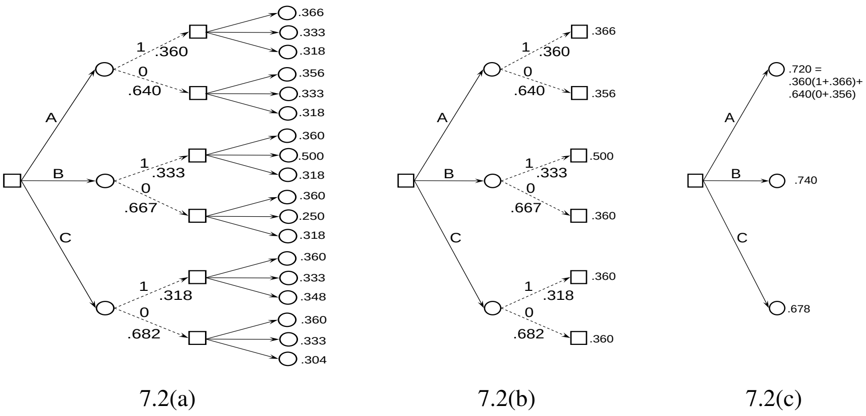


Figure 7.2 (a) Expected value of a hit in the second at-bat; (b) Value of best hitter after one at-bat; (c) Expected value of each hitter before first at-bat.

Although player A initially has the highest batting average, our analysis says that we should try player B for the first at-bat. Why is this? On further examination, we realize that it has a lot to do with the fact that player B has had only three at-bats. If this player gets a hit, our estimate of his probability of getting a hit jumps to 0.500, although it drops to 0.250 if he does not get a hit. If player A gets a hit, his batting average moves from 0.360 to 0.366, reflecting the weight of his much longer record. This is our first hint that it can be useful to collect information about choices where there is the greatest uncertainty.

This example illustrates a setting where observations change our beliefs, which we build into the tree. We could have built our tree where all probabilities remain static, which is typical in decision trees. Imbedding the process of updating probabilities within the decision tree is what distinguishes classical decision trees from the use of decision trees in a learning setting.

Decision trees are actually a powerful strategy for learning, although they have not attracted much attention in the learning literature. One reason is simply that they are computationally more difficult, and for most applications, they do not actually work better. Another is that they are harder to analyze, which makes them less interesting in the research communities that analyze algorithms.

7.3.6 Hybrid policies

It is possible, of course, to use mixed strategies. One of the most popular is known as *epsilon-greedy* exploration. We might specify an exploration rate ϵ where ϵ is the fraction of iterations where decisions should be chosen at random (exploration). The intuitive appeal of this approach is that we maintain a certain degree of forced exploration, while the exploitation steps focus attention on the states that appear to be the most valuable.

In practice, using a mix of exploration steps only adds value for problems with relatively small state or action spaces. The only exception arises when the problem lends itself to an approximation which is characterized by a relatively small number of parameters. Otherwise, performing, say, 1000 exploration steps for a problem with 10^{100} states may provide little or no practical value.

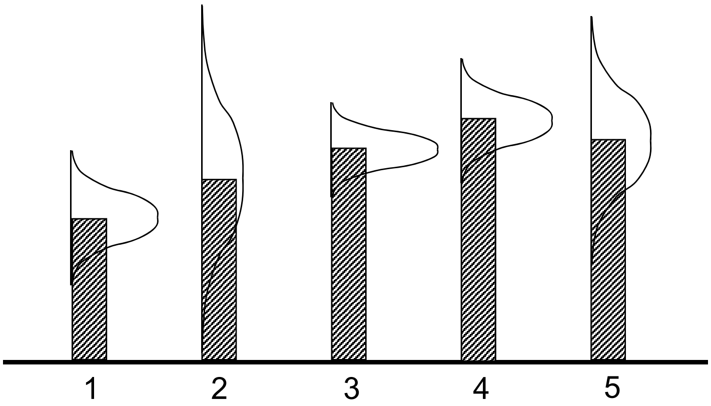


Figure 7.3 A lookup table belief model for five alternatives.

A useful variation is to let ϵ decrease with the number of iterations. For example, let

$$\epsilon^n(s) = c/N^n(s)$$

where $0 < c < 1$ and where $N^n(s)$ is the number of times we have visited state s by iteration n. When we explore, we will choose an action a with probability $1/|\mathcal{A}|$. We choose to explore with probability $\epsilon^n(s)$. This means that the probability we choose decision a when we are in state s, given by $P^n(s,a)$, is at least $\epsilon^n(s)/|\mathcal{A}|$. This guarantees that we will visit every state infinitely often since

$$\sum_{n=1}^{\infty} P^n(s,a) = \sum_{n=1}^{\infty} \epsilon^n(s)/|\mathcal{A}| = \infty.$$

7.3.7 Discussion

Figure 7.3 illustrates a lookup table belief model for five alternatives, where the distribution of belief about each alternative can be computed using frequentist or Bayesian methods. The figure illustrates the type of challenge we face when designing a learning policy. Alternative 4 looks like it is the best, but we have a high level of uncertainty about the second alternative, to the point that it might even be the best. By contrast, we have more confidence in our estimate of alternative 5 than we are for alternative 2, but because the estimate is better, we have to recognize that any of alternatives 2, 4 and 5 might be best.

This figure helps to highlight the challenge of designing a good learning policy. We have to strike a balance between exploring our function to better learn the function, and exploiting the information we have to make good choices. This "exploration vs. exploitation" tradeoff is most apparent when we are maximizing cumulative reward because trying an alternative that does not look as good (but might be the best) may mean that we receive a lower reward, which reduces our performance. This tradeoff, however, is present regardless of whether we are maximizing the cumulative reward or the final reward, although if we are maximizing the final reward the issue only arises when we have a finite budget.

7.4 EVALUATING THE PERFORMANCE OF A POLICY

There are two different ways to evaluate a policy. The first is to look at how well we do in terms of our objective function. The second focuses purely on whether we are choosing the best alternative.

7.4.1 Objective function performance

Below is a list of metrics that have been drawn from different communities.

Empirical performance

We might simulate a policy K times, where each repetition involves making N observations of W. We let ω^k represent a full sample realization of these N observations, which we would denote by $W^1(\omega^k), \ldots, W^N(\omega^k)$. Each sequence ω^k creates a design decision $x^{\pi,N}(\omega^k)$.

It is useful to separate the random variable W that we observe while learning from the random variable we use to evaluate a design, so we are going to let W be the random variable we observe while learning, and we are going to let $\widehat{W}$ be the random variable we use for evaluating a design. Most of the time these are the same random variable with the same distribution, but it opens the door to allowing them to be different.

Once we obtain a design $x^{\pi,N}(\omega^k)$, we then have to evaluate it by taking, say, L observations of $\widehat{W}$, which we designate by $\widehat{W}^1, \ldots, \widehat{W}^\ell, \ldots, \widehat{W}^L$. Using this notation, we would approximate the performance of a design $x^{\pi,N}(\omega^k)$ using

$$\overline{F}^\pi(\omega^k) = \frac{1}{L}\sum_{\ell=1}^{L} F(x^{\pi,N}(\omega^k), \widehat{W}^\ell)$$

We then average over all ω^k using

$$\overline{F}^\pi = \frac{1}{K}\sum_{n=0}^{K-1} \overline{F}^\pi(\omega^k)$$

Quantiles

Instead of evaluating the average performance, we may wish to evaluate a policy based on some quantile. For example, if we are maximizing performance, we might be interested in the $10th$ percentile, since a policy that produces good average performance may work very poorly some of the time.

Let $Q_\alpha(R)$ be the α quantile of a random variable R. Let $F^\pi = F(x^{\pi,N}, W)$ be the random variable describing the performance of policy π, recognizing that we may have uncertainty about the model (captured by S_0), uncertainty in the experiments $W^1, \ldots, W^N$ that go into the final design $x^{\pi,N}$, and then uncertainty in how well we do when we implement $x^{\pi,N}$ due to $\widehat{W}$. Now, instead of taking an expectation of F^π as we did before, we let

$$V^\pi_\alpha = Q_\alpha F(x^{\pi,N}, \widehat{W}).$$

We anticipate that there are many settings where the α quantile is more interesting than an expectation. However, we have to caution that optimizing the α quantile is much harder than optimizing an expectation. This is an important issue when dealing with risk measures, a topic we return to in chapter 21.

Static regret - Deterministic setting

We illustrate static regret for deterministic problems using the context of machine learning where our decision is to choose a parameter θ that fits a model $f(x|\theta)$ to observations y. Here, "x" plays the role of data rather than a decision, although later we will get to "decide" what data to collect (confused yet?).

The machine learning community likes to evaluate the performance of a machine learning algorithm (known as a "learner") which is searching for the best parameters θ to fit some model $f(x|\theta)$ to predict a response y. Imagine a dataset $x^1, \ldots, x^n, \ldots, x^N$ and let $L^n(\theta)$ be the loss function that captures how well our function $f(x^n|\theta^n)$ predicts the response y^{n+1}, where θ^n is our estimate of θ based on the first n observations. Our loss function might be

$$L^{n+1}(x^n, y^{n+1}|\theta^n) = (y^{n+1} - f(x^n|\theta^n))^2.$$

Assume now that we have an algorithm (or policy) for updating our estimate of θ that we designate $\Theta^\pi(S^n)$, where S^n captures whatever the algorithm (or policy) needs to update θ^{n-1} to θ^n. One example of a policy is to optimize over the first n data points, so we would write

$$\Theta^\pi(S^n) = \arg\min_\theta \sum_{m=0}^{n-1} L^{m+1}(x^m, y^{m+1}|\theta)$$

Alternatively, we could use one of the gradient-based algorithms presented in chapter 5. If we fix this policy, our total loss would be

$$L^\pi = \sum_{n=0}^{N-1} L^{n+1}(x^n, y^{n+1}|\Theta^\pi(S^n)).$$

Now imagine that we pick the best value of θ, which we call θ^*, based on all the data. This requires solving

$$L^{static,*} = \min_\theta \sum_{n=0}^{N-1} L^{n+1}(x^n, y^{n+1}|\theta).$$

We now compare the performance of our policy, L^π, to our static bound, $L^{static,*}$. The difference is known as the *static regret* in the machine learning community, or the *opportunity cost* in other fields. The regret (or opportunity cost) is given by

$$R^{static,\pi} = L^\pi - L^{static,*}. \tag{7.22}$$

Static regret - Stochastic setting

Returning to the setting where we have to decide which alternative x to try, we now illustrate static regret in a stochastic setting, where we seek to maximize rewards ("winnings") W_x^n by trying alternative x in the nth trial. Let $X^\pi(S^n)$ be a policy that determines the alternative x^n to evaluate given what we know after n experiments (captured by our state variable S^n). Imagine that we can generate the entire sequence of winnings W_x^n for all alternatives x, and all iterations n. If we evaluate our policy on a single dataset (as we did in the machine learning setting), we would evaluate our regret (also known as *static regret*) as

$$R^{\pi,n} = \max_x \sum_{m=1}^n W_x^m - \sum_{m=1}^n W_{X^\pi(S^m)}^m. \tag{7.23}$$

Alternatively, we could write our optimal solution at time n as

$$x^n = \arg\max_x \sum_{m=1}^n W_x^m,$$

and then write the regret as

$$R^{\pi,n} = \sum_{m=1}^n W_{x^n}^m - \sum_{m=1}^n W_{X^\pi(S^m)}^m.$$

The regret (for a deterministic problems) $R^{\pi,n}$ is comparing the best decision at time n assuming we know all the values $W_x^m,\ x \in \mathcal{X}$ for $m = 1, \ldots, n$, against what our policy $X^\pi(S^m)$ would have chosen given just what we know at time m (please pay special attention to the indexing). This is an instance of static regret for a deterministic problem.

In practice, W_x^m is a random variable. Let $W_x^m(\omega)$ be one sample realization for a sample path $\omega \in \Omega$ (we can think of regret for a deterministic problem as the regret for a single sample path). Here, ω represents a set of all possible realizations of W over all alternatives x, and all iterations n. Think of specifying ω as pre-generating all the observations of W that we *might* experience over all experiments. However, when we make a decision $X^\pi(S^m)$ at time m, we are not allowed to see any of the information that might arrive at times after m.

When we introduce uncertainty, there are now two ways of evaluating regret. The first is to assume that we are going to first observe the outcomes $W_x^m(\omega)$ for all the alternatives and the entire history $m = 1, \ldots, n$, and compare this to what our policy $X^\pi(S^m)$ would have done at each time m knowing only what has happened up to time m. The result is the regret for a single sample path ω

$$R^{\pi,n}(\omega) = \max_{x(\omega)} \sum_{m=1}^n W_{x(\omega)}^m(\omega) - \sum_{m=1}^n W_{X^\pi(S^m)}^m(\omega). \tag{7.24}$$

As we did above, we can also write our optimal decision for the stochastic case as

$$x^n(\omega) = \arg\max_{x \in \mathcal{X}} \sum_{m=1}^n W_x^m(\omega).$$

We would then write our regret for sample path ω as

$$R^{\pi,n}(\omega) = \sum_{m=1}^n W_{x^n(\omega)}^m(\omega) - \sum_{m=1}^n W_{X^\pi(S^m)}^m(\omega).$$

Think of $x^n(\omega)$ as the best answer if we actually did know $W_x^m(\omega)$ for $m = 1, \ldots, n$, which in practice would never be true.

If we use our machine learning setting, the sample ω would be a single dataset used to fit our model. In machine learning, we typically have a single dataset, which is like working with a single ω. This is typically what is meant by a deterministic problem (think about it). Here, we are trying to design policies that will work well across many datasets.

In the language of probability, we would say that $R^{\pi,n}$ is a random variable (since we would get a different answer each time we run the simulation), while $R^{\pi,n}(\omega)$ is a sample realization. It helps when we write the argument (ω) because it tells us what is random, but $R^{\pi,n}(\omega)$ and $x^n(\omega)$ are sample realizations, while $R^{\pi,n}$ and x^n are considered random variables (the notation does not tell you that they are random - you just have to know it). We can "average" over all the outcomes by taking an expectation, which would be written

$$\mathbb{E}R^{\pi,n} = \mathbb{E}\left\{W^n_{x^n} - \sum_{m=1}^{n} W^m_{X^\pi(S^m)}\right\}.$$

Expectations are mathematically pretty, but we can rarely actually compute them, so we run simulations and take an average. Assume we have a set of sample realizations $\omega \in \hat{\Omega} = \{\omega^1, \ldots, \omega^\ell, \ldots, \omega^L\}$. We can compute an average regret (approximating expected regret) using

$$\mathbb{E}R^{\pi,n} \approx \frac{1}{L}\sum_{\ell=1}^{L} R^{\pi,n}(\omega^\ell).$$

Classical static regret assumes that we are allowed to find a solution $x^n(\omega)$ for each sample path. There are many settings where we have to find solutions before we see any data, that works well, on average, over all sample paths. This produces a different form of regret known in the computer science community as *pseudo-regret* which compares a policy $X^\pi(S^n)$ to the solution x^* that works best *on average* over all possible sample paths. This is written

$$\bar{R}^{\pi,n} = \max_x \mathbb{E}\left\{\sum_{m=1}^{n} W^n_x\right\} - \mathbb{E}\left\{\sum_{m=1}^{n} W^n_{X^\pi(S^n)}(\omega)\right\}. \tag{7.25}$$

Again, we will typically need to approximate the expectation using a set of sample paths $\hat{\Omega}$.

Dynamic regret

A criticism of static regret is that we are comparing our policy to the best decision x^* (or best parameter θ^* in a learning problem) for an entire dataset, but made after the fact with perfect information. In online settings, it is necessary to make decisions x^n (or update our parameter θ^n) using only the information available up through iteration n.

Dynamic regret raises the bar by choosing the best value θ^n that minimizes $L^n(x^{n-1}, y^n|\theta)$, which is to say

$$\begin{aligned} \theta^{*,n} &= \arg\min_\theta L^n(x^{n-1}, y^n|\theta), && (7.26)\\ &= \arg\min_\theta (y^n - f(x^{n-1}|\theta))^2. && (7.27)\end{aligned}$$

The dynamic loss function is then

$$L^{dynamic,*} = \sum_{n=0}^{N-1} L^{n+1}(x^n, y^{n+1}|\theta^{*,n}).$$

More generally, we could create a policy Θ^π for adaptively evolving θ (equation (7.27) is an example of one such policy). In this case we would compute θ using $\theta^n = \Theta^\pi(S^n)$, where S^n is our belief state at time n (this could be current estimates, or the entire history of data). We might then write our dynamic loss problem in terms of finding the best policy Θ^π for adaptively searching for θ as

$$L^{dynamic,*} = \min_{\Theta^\pi} \sum_{n=0}^{N-1} L^{n+1}(x^n, y^{n+1}|\Theta^\pi(S^n)).$$

We then define dynamic regret using

$$R^{dynamic,\pi} = L^\pi - L^{dynamic,*}.$$

Dynamic regret is simply a performance metric using a more aggressive benchmark. It has attracted recent attention in the machine learning community as a way of developing theoretical benchmarks for evaluating learning policies.

Opportunity cost (stochastic)

Opportunity cost is a term used in the learning community that is the same as regret, but often used to evaluate policies in a stochastic setting. Let $\mu_x = \mathbb{E}F(x,\theta)$ be the true value of design x, let

$$\begin{aligned} x^* &= \arg\max_x \mu_x, \\ x^\pi &= \arg\max_x \mu_{x^{\pi,N}}. \end{aligned}$$

So, x^* is the best design if we knew the truth, while $x^{\pi,N}$ is the design we obtained using learning policy π after exhausting our budget of N experiments. In this setting, μ_x is treated deterministically (think of this as a known truth), but $x^{\pi N}$ is random because it depends on a noisy experimentation process. The expected regret, or opportunity cost, of policy π is given by

$$R^\pi = \mu_{x^*} - \mathbb{E}\mu_{x^{\pi,N}}. \tag{7.28}$$

Competitive analysis

A strategy that is popular in the field known as *online computation* likes to compare the performance of a policy to the best that could have been achieved. There are two ways to measure "best." The most common is to assume we know the future. Assume we are making decisions $x_0, x_1, \ldots, x_T$ over our horizon $0, \ldots, T$. Let ω represent a sample path $W_1(\omega), \ldots, W_T(\omega)$, and let $x_t^*(\omega)$ be the best decision given that we know that all random outcomes (over the entire horizon) are known (and specified by ω). Finally, let $F(x_t, W_{t+1}(\omega))$ be the performance that we observe at time $t+1$. We can then create a perfect foresight (PF) policy using

$$X_t^{PF}(\omega) = \arg\max_{x_t(\omega)} \left(c_t x_t(\omega) + \max_{x_{t+1}(\omega),\ldots,x_T(\omega)} \sum_{t'=t+1}^{T} c_{t'} x_{t'}(\omega) \right).$$

Unlike every other policy that we consider in this volume, this policy is allowed to see into the future, producing decisions that are better than anything we could achieve without this ability. Now consider some $X^\pi(S_t)$ policy that is only allowed to see the state at time S_t. We can compare policy $X^\pi(S)$ to our perfect foresight using the *competitive ratio* given by

$$\rho^\pi = \mathbb{E}\frac{\sum_{t=0}^{T-1} F(X_t^\pi(\omega), W_{t+1}(\omega))}{\sum_{t=0}^{T-1} F(X_t^{PF}(\omega), W_{t+1}(\omega))}$$

where the expectation is over all sample paths ω (competitive analysis is often performed for a single sample path). Researchers like to prove bounds on the competitive ratio, although these bounds are never tight.

Indifference zone selection

A variant of the goal of choosing the best alternative $x^* = \arg\max_x \mu_x$ is to maximize the likelihood that we make a choice $x^{\pi,N}$ that is almost as good as x^*. Assume we are equally happy with any outcome within δ of the best, by which we mean

$$\mu_{x^*} - \mu_{x^{\pi,N}} \leq \delta.$$

The region $(\mu_{x^*} - \delta, \mu_{x^*})$ is referred to as the *indifference zone*. Let $V^{n,\pi}$ be the value of our solution after n experiments. We require $\mathbb{P}^\pi\{\mu_{d^*} = \bar{\mu}^*|\mu\} > 1 - \alpha$ for all μ where $\mu_{[1]} - \mu_{[2]} > \delta$, and where $\mu_{[1]}$ and $\mu_{[2]}$ represent, respectively, the best and second best choices.

We might like to maximize the likelihood that we fall within the indifference zone, which we can express using

$$P^{IZ,\pi} = \mathbb{P}^\pi(V^{\pi,n} > \mu^* - \delta).$$

As before, the probability has to be computed with the appropriate Bayesian or frequentist distribution.

7.4.2 Finding the best alternative

A second set of performance metrics is based on our ability to choose the best alternative in the set $\mathcal{X} = \{x_1, \ldots, x_M\}$:

Asymptotic convergence for terminal reward

While in practice we need to evaluate how an algorithm does in a finite budget, there is a long tradition in the analysis of algorithms to study the asymptotic performance of algorithms when using a final-reward criterion. In particular, if x^* is the solution to our asymptotic formulation in equation (7.1), we would like to know if our policy that produces a solution $x^{\pi,N}$ after N evaluations would eventually converge to x^*. That is, we would like to know if

$$\lim_{N\to\infty} x^{\pi,N} \to x^*.$$

Researchers will often begin by proving that an algorithm is asymptotically convergent (as we did in chapter 5, and then evaluate the performance in a finite budget N empirically. Asymptotic analysis generally only makes sense when using a final-reward objective.

Finite time bounds on choosing the wrong alternative

There is a body of research that seeks to bound the number of times a policy chooses a suboptimal alternative (where alternatives are often referred to as "arms" for a multiarmed bandit problem). Let μ_x be the (unknown) expected reward for alternative x, and let $W_x^n = \mu_x + \epsilon_x^n$ be the observed random reward from trying x. Let x^* be the optimal alternative, where

$$x^* = \arg\max_x \mu_x.$$

For these problems, we would define our loss function as

$$L^n(x^n) = \begin{cases} 1 & \text{If } x^n \neq x^*, \\ 0 & \text{Otherwise.} \end{cases}$$

Imagine that we are trying to minimize the cumulative reward, which means the total number of times that we do not choose the best alternative. We can compare a policy that chooses $x^n = X^\pi(S^n)$ against a perfect policy that chooses x^* each time. The regret for this setting is then simply

$$R^{\pi,n} = \sum_{m=1}^{n} L^n(X^\pi(S^n)).$$

Not surprisingly, R^π grows monotonically in n, since good policies have to be constantly experimenting with different alternatives. An important research goal is to design bounds on $R^{\pi,n}$, which is called a finite-time bound, since it applies to $R^{\pi,n}$ for finite n.

Probability of correct selection

A different perspective is to focus on the probability that we have selected the best out of a set $\mathcal{X}$ alternatives. In this setting, it is typically the case that the number of alternatives is not too large, say 10 or 20, and certainly not 100,000. Assume that

$$x^* = \arg\max_{x\in\mathcal{X}} \mu_x$$

is the best decision (for simplicity, we are going to ignore the presence of ties). After n samples, we would make the choice

$$x^n = \arg\max_{x\in\mathcal{X}} \bar{\mu}_x^n.$$

This is true regardless of whether we are using a frequentist or Bayesian estimate.

We have made the correct selection if $x^n = x^*$, but even the best policy cannot guarantee that we will make the best selection every time. Let $1_{\{\mathcal{E}\}} = 1$ if the event $\mathcal{E}$ is true, 0 otherwise. We write the probability of correct selection as

$$\begin{aligned} P^{CS,\pi} &= \text{probability we choose the best alternative} \\ &= \mathbb{E}^\pi 1_{\{x^n = x^*\}}, \end{aligned}$$

where the underlying probability distribution depends on our experimental policy π. The probability is computed using the appropriate distribution, depending on whether we are using Bayesian or frequentist perspectives. This may be written in the language of loss functions. We would define the loss function as

$$L^{CS,\pi} \quad = \quad 1_{\{x^n \neq x^*\}}.$$

Although we use $L^{CS,\pi}$ to be consistent with our other notation, this is more commonly represented as L_{0-1} for "0-1 loss."

Note that we write this in terms of the negative outcome so that we wish to minimize the loss, which means that we have not found the best selection. In this case, we would write the probability of correct selection as

$$P^{CS,\pi} = 1 - \mathbb{E}^{\pi} L^{CS,\pi}.$$

Subset selection

Ultimately our goal is to pick the best design. Imagine that we are willing to choose a subset of designs S, and we would like to ensure that $P(x^* \in S) \geq 1 - \alpha$, where $1/|\mathcal{X}| < 1 - \alpha < 1$. Of course, it would be idea if $|\mathcal{S}| = 1$ or, failing this, as small as possible. Let $\bar{\mu}^n_x$ be our estimate of the value of x after n experiments, and assume that all experiments have a constant and known variance σ. We include x in the subset if

$$\bar{\mu}^n_x \geq \max_{x' \neq x} \bar{\mu}^n_{x'} - h\sigma\sqrt{\frac{2}{n}}.$$

The parameter h is the $1 - \alpha$ quantile of the random variable $\max_i Z^n_i$ where Z^n_i is given by

$$Z^n_i = \frac{(\bar{\mu}^n_i - \bar{\mu}^n_x) - (\mu_i - \mu_x)}{\sigma\sqrt{2/n}}.$$

7.5 LEARNING AND THE MULTIARMED BANDIT PROBLEM

We would not be doing justice to the learning literature if we did not acknowledge the contribution of a substantial body of literature that addresses what is known as the *multiarmed bandit problem*. The term comes from the common description (in the United States) that a slot machine (in American English), which is sometimes known as a "fruit machine" (in British English), is a "one armed bandit" since each time you pull the arm on the slot machine you are likely to lose money.

Now imagine that you have to choose which out of a group of slot machines to play (a surprising fiction since winning probabilities on slot machines are carefully calibrated). Imagine (and this is a stretch) that each slot machine has a different winning probability, and that the only way to learn about the winning probability is to play the machine and observe the winnings. This may mean playing a machine where your estimate of winnings is low, but you acknowledge that your estimate may be wrong, and that you have to try playing the machine to improve your knowledge.

This classic problem has several notable characteristics. The first and most important is the tradeoff between exploration (trying an arm that does not seem to be the best in order to learn more about it) and exploitation (trying arms with higher estimated winnings in order to maximize winnings over time), where winnings are accumulated over time. Other distinguishing characteristics: discrete choices (that is, slot machines), beliefs about each individual machine, and an underlying process that is stationary (the distribution of winnings does not change over time).

Before proceeding, we need to first pause and emphasize that the "multiarmed bandit problem" is simply derivative-free stochastic optimization where we are trying to maximize cumulative reward (although there are "bandit" papers that focus on terminal reward, but these are rare in this community). While it is important to recognize the relationship of bandit problems to the broader stochastic optimization literature, it is also important to recognize the unique contributions of what is known as the "bandit community."

Multiarmed bandit problems first attracted the attention of the applied probability community in the 1950's, initially in the context of the simpler two-armed problem. The multiarmed problem resisted computational solution until the development in 1974 by J.C. Gittins who identified a novel decomposition that led to what are known as *index policies* which involves computing a value ("index") for each arm, and then choosing the arm with the greatest index. While "Gittins indices" (as they came to be known) remain computationally difficult to compute which has limited their widespread adoption, the elegant simplicity of index policies has guided research into an array of policies that are quite practical.

In 1985, a second breakthrough came from the computer science community, when it was found that a very simple class of policies known as *upper confidence bound* policies (also described below) enjoyed nice theoretical properties in the form of bounds on the number of times that the wrong arm would be visited. The ease with which these policies can be computed (they are a form of index policy) has made them particularly popular in high speed settings such as the internet where there are many situations where it is necessary to make good choices, such as which ad to post to maximize the value of an array of services.

Today, the literature on "bandit problems" has expanded far from its original roots to include any sequential learning problem (which means the state S^n includes a belief state about the function $\mathbb{E}F(x, W)$) where we control the decisions of where to evaluate $F(x, W)$. However, bandit problems now include many problem variations, such as

- Maximizing the terminal reward rather than just cumulative rewards.
- "Arms" no longer have to be discrete; x may be continuous and vector-valued.
- Instead of one belief about each arm, a belief might be in the form of a linear model that depends on features drawn from x.

The bandit community fostered a culture of creating problem variations, and then deriving index policies or proving bounds on UCB policies. While the actual performance of the UCB policies requires careful experimentation and tuning, the culture of creating problem variations is unparalleled in the stochastic optimization literature. Table 7.2 lists a sampling of these bandit problems, with the original multiarmed bandit problem at the top.

7.6 GITTINS INDICES FOR LEARNING WITH CUMULATIVE REWARDS

We address the problem of learning the value of $\mu_x = \mathbb{E}F(x, W)$ for discrete x by trying different alternatives and observing $W_x^n = \mu_x + \varepsilon_x^n$. The challenge is to find a policy

Bandit problem	Description
Multiarmed bandits	Basic problem with discrete alternatives, online (cumulative regret) learning, lookup table belief model with independent beliefs
Restless bandits	Truth evolves exogenously over time
Adversarial bandits	Distributions from which rewards are being sampled can be set arbitrarily by an adversary
Continuum-armed bandits	Arms are continuous
X-armed bandits	Arms are a general topological space
Contextual bandits	Exogenous state is revealed which affects the distribution of rewards
Dueling bandits	The agent gets a relative feedback of the arms as opposed to absolute feedback
Arm-acquiring bandits	New machines arrive over time
Intermittent bandits	Arms are not always available
Response surface bandits	Belief model is a response surface (typically a linear model)
Linear bandits	Belief is a linear model
Dependent bandits	A form of correlated beliefs
Finite horizon bandits	Finite-horizon form of the classical infinite horizon multi-armed bandit problem
Parametric bandits	Beliefs about arms are described by a parametric belief model
Nonparametric bandits	Bandits with nonparametric belief models
Graph-structured bandits	Feedback from neighbors on graph instead of single arm
Extreme bandits	Optimize the maximum of recieved rewards
Quantile-based bandits	The arms are evaluated in terms of a specified quantile
Preference-based bandits	Find the correct ordering of arms
Best-arm bandits	Identify the optimal arm with the largest confidence given a fixed budget

Table 7.2 A sample of the growing population of “bandit” problems.

$X^\pi(S^n)$ that solves

$$\max_\pi \mathbb{E} \sum_{n=0}^{\infty} \gamma^n W^{n+1}_{X^\pi(S^n)},$$

where $x^n = X^\pi(S^n)$ and where γ is a discount factor that satisfies $0 \leq \gamma < 1$.

The first implementable and provably optimal policy for this problem class was developed by John Gittins in 1974, with a strategy that became known as *Gittins indices*. The idea is to compute a single value (called an index) for each alternative x, and then evaluate the alternative with the highest index. The approach is based on solving Bellman's equation, which puts Gittins indices in the class of policies based on value functions (we referred to these as VFA policies above).

7.6.1 Foundations

Let $\mathcal{X}$ be the set of alternatives ("arms" in the language of the bandit community), and let W_x be the random variable that gives the amount that we win if we play arm x. We can think of W_x as a sampled realization of $\mathbb{E}F(x, W)$, where we assume we know the function $F(x, W)$ but need to observe W. However, we will often use the convention that we just observe W and treat this as our observation of the performance.

The theory surrounding bandit problems, and much of the literature on information collection, has evolved in a Bayesian framework where we view μ_x, the true value of alternative x, to be a random variable. We assume we begin with a prior distribution of belief where μ_x is normally distributed with mean $\bar{\mu}_x^0$ and precision β_x^0 (recall that this is the inverse of the variance).

We next need to model how an observation changes our distribution of belief.

After n experiments, assume that our belief has evolved to where our belief about μ_x is normally distributed with mean $\bar{\mu}_x^n$ and precision β_x^n. We write our *belief state* as $S^n = (\bar{\mu}_x^n, \beta_x^n)_{x\in\mathcal{X}}$, with the assumption of normality implicit (this is actually captured in the initial state S_0). Assume that we choose to evaluate $x^n = X^\pi(S^n)$, after which we observe $W^{n+1} = W^{n+1}$, where we assume that the precision of our observation W^{n+1} of the function is known and given by β^W, if it does not depend on x, or β_x^W if it does depend on x. We would then use this information to update our belief using our Bayesian updating formulas (first presented in section 3.4), given by

$$\bar{\mu}_x^{n+1} = \begin{cases} \frac{\beta_x^n \bar{\mu}_x^n + \beta_x^W W_x^{n+1}}{\beta_x^n + \beta_x^W} & \text{if } x = x^n \\ \bar{\mu}_x^n & \text{otherwise,} \end{cases} \tag{7.29}$$

$$\beta_x^{n+1} = \begin{cases} \beta_x^n + \beta_x^W & \text{if } x = x^n \\ \beta_x^n & \text{otherwise.} \end{cases} \tag{7.30}$$

Our goal is to find a policy that determines which action to take to collect information. Let $x^n = X^\pi(S^n)$ be the action we take after making n observations using policy π. We can state the problem of finding the best policy in terms of solving

$$\max_\pi \mathbb{E} \sum_{n=0}^{\infty} \gamma^n \mu_{x^n},$$

where $x^n = X^\pi(S^n)$, and where γ is a discount factor with $0 \le \gamma < 1$. We can equivalently write this as

$$\max_\pi \mathbb{E} \sum_{n=0}^{\infty} \gamma^n \bar{\mu}_{x^n}^n,$$

keeping in mind that x^n must depend on the information in S^n.

One way to solve the problem is to use Bellman's equation

$$V^n(S^n) = \max_{x\in\mathcal{X}} \big(C(S^n, x) + \gamma \mathbb{E}\{V^{n+1}(S^{n+1})|S^n, x\}\big), \tag{7.31}$$

It is important to keep in mind that S^n is our belief state, and that $V^n(S^n)$ is the expected value of our earnings given our current state of belief. The problem is that if we have $|\mathcal{X}|$ actions, then our state variable has $2|\mathcal{X}|$ continuous dimensions (the mean and variance for each alternative).

The recognition that learning problems could be formulated as dynamic programs around the belief state represented an important development in the learning community. Even Bellman (the father of dynamic programming whose work is reviewed in greater depth in chapter 14) would refer to belief states as “hyperstates” without realizing that there is no difference between information about inventories, information about the weather, and information about probability distributions. The real breakthrough with recognizing that Bellman’s equation (7.31) was that it was the first characterization of an optimal policy for this problem class. The challenge was that (7.31) is computationally intractable, since, for the case of normally distributed rewards, the belief state S^n has two continuous dimensions (the mean and variance) for each arm.

In a landmark paper (Gittins & Jones (1974)), it was shown that equation (7.31) could be reduced to a single dynamic program for each individual arm. This means that we now only need to solve (7.31) for a two-dimensional problem, which can be done using numerical integration (this does not mean it is easy - it just means it can be done). Each of these single-armed dynamic programs produces an index ν^n_x, where the optimal policy involves finding the largest ν^n_x for all $x \in \mathcal{X}$. While the resulting policy is still much more difficult to solve than other policies we will introduce, it is a rare instance of a truly optimal policy, and provides insights that guide the design of effective, but simpler, learning policies.

7.6.2 Basic theory of Gittins indices

Assume we face the choice of playing a single slot machine, or stopping and converting to a process that pays a fixed reward ρ in each time period until infinity. If we choose to stop sampling and accept the fixed reward, the total future reward is $\rho/(1-\gamma)$. Alternatively, if we play the slot machine, we not only win a random amount W, we also learn something about the parameter μ that characterizes the distribution of W, where $\mu = \mathbb{E}W$. After n observations of W, we let S^n capture the parameters of the probability distribution of μ given what we have learned. This distribution depends on the nature of the underlying uncertainty, such as:

Normally distributed If we have a normally distributed prior on μ, and W is normally distributed, then our belief about μ after n observations is also normally distributed with distribution $N(\bar{\mu}^n, \bar{\sigma}^{2,n})$, which means $S^n = (\bar{\mu}^n, \bar{\sigma}^{2,n})$ which is updated using equations (7.11) - (7.12).

Binomial distribution Imagine that μ is 0 or 1, and let $\mathbf{p}$ be the probability that $\mu = 1$, and that $W \in \{0.1\}$. Let $\bar{p}^n$ be our best estimate of $\mathbf{p}$, which represents our distribution of belief about μ. Our state is given by $S^n = \bar{p}^n$ which is updated using

$$\bar{p}^{n+1} = \left(1 - \frac{1}{n+1}\right)\bar{p}^n + \frac{1}{n+1}W^{n+1}.$$

(We could also model the problem where the state is $N^n = \sum_{i=1}^n W^i$.)

Bernoulli distribution Finally assume that we may win a game with some probability $\mathbf{p}$, where now we are interested in finding the true probability. We can represent our distribution of belief about the random variable $\mathbf{p}$ using the beta distribution given by

$$f(p|\alpha,\beta) = \begin{cases} \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} p^{\alpha-1}(1-p)^{\beta-1} & \text{if } 0 < p < 1 \\ 0 & \text{otherwise.} \end{cases}$$

which is characterized by the parameters α and β, so we would write $S^n = (\alpha^n, \beta^n)$ which are updated using

$$\begin{aligned} \alpha^{n+1} &= \alpha^n + W^{n+1}, \\ \beta^{n+1} &= \beta^n + \left(1 - W^{n+1}\right). \end{aligned}$$

The estimate S^n represents our state variable, which is to say our distribution of belief about μ. Let $\bar{\mu}^n$ be our current estimate of the reward we would receive from playing the machine given what we know after n plays. The optimality equations can now be written

$$V(S^n, \rho) = \max\left[\rho + \gamma V(S^n, \rho), \bar{\mu}^n + \gamma \mathbb{E}\left\{ V(S^{n+1}, \rho) \middle| S^n \right\}\right], \qquad (7.32)$$

where S^{n+1} is computed from S^n and W^{n+1} using one of the updating equations illustrated in the examples above. We use the notation $V(S^n, \rho)$ to express the dependence on ρ.

Since we have an infinite horizon problem, the value function must satisfy the optimality equations

$$V(S, \rho) = \max\left[\rho + \gamma V(S, \rho), \mu + \gamma \mathbb{E}\left\{ V(S', \rho) | S \right\}\right],$$

where S' is the updated state, and μ is the mean that is captured in the state S.

It can be shown that if we choose to stop sampling in iteration n and accept the fixed payment ρ, then that is the optimal strategy for all future rounds. This means that starting at iteration n, our optimal future payoff (once we have decided to accept the fixed payment) is

$$\begin{aligned} V(S, \rho) &= \rho + \gamma\rho + \gamma^2\rho + \cdots \\ &= \frac{\rho}{1-\gamma}, \end{aligned}$$

which means that we can write our optimality recursion in the form

$$V(S^n, \rho) = \max\left[\frac{\rho}{1-\gamma}, \bar{\mu}^n + \gamma \mathbb{E}\left\{ V(S^{n+1}, \rho) \middle| S^n \right\}\right]. \qquad (7.33)$$

Now for the magic of Gittins indices. Let ν be the value of ρ which makes the two terms in the brackets in (7.33) equal. That is,

$$\frac{\nu}{1-\gamma} = \mu + \gamma \mathbb{E}\left\{ V(S, \nu) | S \right\}. \qquad (7.34)$$

We are going to proceed for the case of normally distributed beliefs and observations. We assume that W is random with a known variance σ_W^2. Let $\nu^{Gitt}(\mu, \sigma, \sigma_W, \gamma)$ be the solution of (7.34). The optimal solution depends on the current estimate of the mean, μ, its variance σ^2, the variance of our measurements σ_W^2, and the discount factor γ. (For notational simplicity, we are assuming that the experimental noise σ_W^2 is independent of the action x, but this assumption is easily relaxed.)

Next assume that we have a family of slot machines $\mathcal{X}$, and let $\nu_x^{Gitt,n}(\bar{\mu}_x^n, \bar{\sigma}_x^n, \sigma_W, \gamma)$ be the value of ν that we compute for each slot machine $x \in \mathcal{X}$ given state $S^n = (\bar{\mu}_x^n, \bar{\sigma}_x^n)_{x \in \mathcal{X}}$. An optimal policy for selecting slot machines is to choose the slot machine with the highest value for $\nu_x^{Gitt,n}(\bar{\mu}_x^n, \bar{\sigma}_x^n, \sigma_W, \gamma)$. Such policies are known as *index policies*, which refers to the property that the parameter $\nu_x^{Gitt,n}(\bar{\mu}_x^n, \bar{\sigma}_x^n, \sigma_W, \gamma)$ for alternative x depends only on the characteristics of alternative x. For this problem, the parameters $\nu_x^{Gitt,n}(\bar{\mu}_x^n, \bar{\sigma}_x^n, \sigma_W, \gamma)$ are called Gittins indices.

When we ignore the value of acquiring information, we would make a decision by solving

$$\max_x \bar{\mu}_x^n.$$

This is known as a pure exploitation policy, and might easily lead us to avoid a decision that might be quite good, but which we currently think is poor (and we are unwilling to learn anything more about the decision). Gittins theory tells us to solve

$$\max_x \nu_x^{Gitt,n}(\bar{\mu}_x^n, \bar{\sigma}_x^n, \sigma_W, \gamma).$$

The beauty of Gittins indices (or any index policy) is that it reduces N-dimensional problems into a series of one-dimensional problems. The problem is that solving equation (7.33) (or equivalently, (7.34)) offers its own challenges. Finding $\nu_x^{Gitt,n}(\mu, \sigma, \sigma_W, \gamma)$ requires solving the optimality equation in (7.33) for different values of ρ until (7.34) is satisfied. Although algorithmic procedures have been designed for this, they are not simple.

7.6.3 Gittins indices for normally distributed rewards

Students learn in their first statistics course that normally distributed random variables satisfy a nice property. If Z is normally distributed with mean 0 and variance 1 and if

$$X = \mu + \sigma Z$$

then X is normally distributed with mean μ and variance σ^2. This property simplifies what are otherwise difficult calculations about probabilities of events.

The same property applies to Gittins indices. Although the proof requires some development, it is possible to show that

$$\nu^{Gitt,n}(\bar{\mu}^n, \bar{\sigma}^n, \sigma_W, \gamma) = \mu + \Gamma(\frac{\bar{\sigma}^n}{\sigma_W}, \gamma)\sigma_W,$$

where

$$\Gamma(\frac{\bar{\sigma}^n}{\sigma_W}, \gamma) = \nu^{Gitt,n}(0, \sigma, 1, \gamma)$$

is a "standard normal Gittins index" for problems with mean 0 and variance 1. Note that $\bar{\sigma}^n/\sigma_W$ decreases with n, and that $\Gamma(\frac{\bar{\sigma}^n}{\sigma_W}, \gamma)$ decreases toward zero as $\bar{\sigma}^n/\sigma_W$ decreases. As $n \rightarrow \infty$, $\nu^{Gitt,n}(\bar{\mu}^n, \bar{\sigma}^n, \sigma_W, \gamma) \rightarrow \bar{\mu}^n$.

Unfortunately, as of this writing, there do not exist easy-to-use software utilities for computing standard Gittins indices. Table 7.3 is exactly such a table for Gittins indices. The table gives indices for both the variance-known and variance-unknown cases, but only for the case where $\frac{\sigma^n}{\sigma_W} = \frac{1}{n}$. In the variance-known case, we assume that σ^2 is given,

	Discount factor			
	Known variance		Unknown variance	
Observations	0.95	0.99	0.95	0.99
1	0.9956	1.5758	-	-
2	0.6343	1.0415	10.1410	39.3343
3	0.4781	0.8061	1.1656	3.1020
4	0.3878	0.6677	0.6193	1.3428
5	0.3281	0.5747	0.4478	0.9052
6	0.2853	0.5072	0.3590	0.7054
7	0.2528	0.4554	0.3035	0.5901
8	0.2274	0.4144	0.2645	0.5123
9	0.2069	0.3808	0.2353	0.4556
10	0.1899	0.3528	0.2123	0.4119
20	0.1058	0.2094	0.1109	0.2230
30	0.0739	0.1520	0.0761	0.1579
40	0.0570	0.1202	0.0582	0.1235
50	0.0464	0.0998	0.0472	0.1019
60	0.0392	0.0855	0.0397	0.0870
70	0.0339	0.0749	0.0343	0.0760
80	0.0299	0.0667	0.0302	0.0675
90	0.0267	0.0602	0.0269	0.0608
100	0.0242	0.0549	0.0244	0.0554

Table 7.3 Gittins indices $\Gamma(\frac{\sigma^n}{\sigma_W}, \gamma)$ for the case of observations that are normally distributed with mean 0, variance 1, and where $\frac{\sigma^n}{\sigma_W} = \frac{1}{n}$ (from Gittins (1989)).

which allows us to calculate the variance of the estimate for a particular slot machine just by dividing by the number of observations.

Lacking standard software libraries for computing Gittins indices, researchers have developed simple approximations. As of this writing, the most recent of these works as follows. First, it is possible to show that

$$\Gamma\left(s,\gamma\right) = \sqrt{-\log\gamma} \cdot b\left(-\frac{s^2}{\log\gamma}\right). \tag{7.35}$$

A good approximation of $b(s)$, which we denote by $\tilde{b}(s)$, is given by

$$\tilde{b}(s) = \begin{cases} \frac{s}{\sqrt{2}} & s \leq \frac{1}{7}, \\ e^{-0.02645(\log s)^2 + 0.89106 \log s - 0.4873} & \frac{1}{7} < s \leq 100, \\ \sqrt{s}\,(2\log s - \log\log s - \log 16\pi)^{\frac{1}{2}} & s > 100. \end{cases}$$

Thus, the approximate version of (7.35) is

$$\nu^{Gitt,n}(\mu, \sigma, \sigma_W, \gamma) \approx \bar{\mu}^n + \sigma_W \sqrt{-\log\gamma} \cdot \tilde{b}\left(-\frac{\bar{\sigma}^{2,n}}{\sigma_W^2 \log\gamma}\right). \tag{7.36}$$

7.6.4 Comments

While Gittins indices was considered a major breakthrough, it has largely remained an area of theoretical interest in the applied probability community. Some issues that need to be kept in mind when using Gittins indices are:

- While Gittins indices were viewed as a computational breakthrough, they are not, themselves, easy to compute.

- Gittins index theory only works for infinite horizon, discounted, cumulative reward problems. Gittins indices are not optimal for finite horizon problems which is what we always encounter in practice, but Gittins indices may still be a useful approximation.

- Gittins theory is limited primarily to lookup table belief models with independent beliefs.

We note that while Gittins indices are not optimal in practical settings, we are not going to introduce any other policy that is optimal; this is a field where computational restrictions limit our attention to suboptimal policies, where we try to prove, either theoretically or experimentally, that particular policies exhibit specific advantages over others.

7.7 THE KNOWLEDGE GRADIENT FOR INDEPENDENT BELIEFS

The knowledge gradient belongs to the value of value-of-information policies which choose alternatives based on the improvement in the quality of the objective from better decisions that arise from a better understanding of the problem. The knowledge gradient works from a Bayesian belief model where our belief about the truth is represented by a probability distribution of possible truths. The basic knowledge gradient calculates the value of a single experiment, but this can be used as a foundation for variations that allow for repeated experiments.

The knowledge gradient was originally developed for offline (terminal reward) settings, so we begin with this problem class. Our experience is that the knowledge gradient is particularly well suited for settings where experiments (or observations) are expensive. For example:

- An airline wants to know the effect of allowing additional schedule slack, which can only be evaluated by running dozens of simulations to capture the variability due to weather. Each simulation may take several hours to run.

- A scientist needs to evaluate the effect of increasing the temperature of a chemical reaction or the strength of a material. A single experiment may take several hours, and needs to be repeated to reduce the effect of the noise in each experiment.

- A drug company is running clinical trials on a new drug, where it is necessary to test the drug at different dosages for toxicity. It takes several days to assess the effect of the drug at a particular dosage.

After developing the knowledge gradient for offline (terminal reward) settings, we show how to compute the knowledge gradient for online (cumulative reward) problems. We begin by discussing belief models, but devote the rest of this section to handling the special case of independent beliefs. Section 7.8 extends the knowledge gradient to several more general belief models.

7.7.1 The belief model

The knowledge gradient uses a Bayesian belief model where we begin with a prior on $\mu_x = \mathbb{E}F(x, W)$ for $x \in \{x_1, \ldots, x_M\}$. We are going to illustrate the key ideas using a lookup table belief model (which is to say, we have an estimate for each value of x), where we initially assume the beliefs are independent. This means that anything we learn about some alternative x does not teach us anything about an alternative x'.

We assume that we believe that the true value of μ_x is described by a normal distribution $N(\bar{\mu}_x^0, \sigma_x^{2,0})$, known as the prior. This may be based on prior experience (such as past experience with the revenue from charging a price x for a new book), some initial data, or from an understanding of the physics of a problem (such as the effect of temperature on the conductivity of a metal).

It is possible to extend the knowledge gradient to a variety of belief models. A brief overview of these is:

Correlated beliefs Alternatives x may be related, perhaps because they are discretizations of a continuous parameter (such as temperature or price), so that μ_x and μ_{x+1} are close to each other. Trying x then teaches us something about μ_{x+1}. Alternatively, x and x' may be two drugs in the same class, or a product with slightly different features. We capture these relationships with a covariance matrix Σ^0 where $\Sigma^0_{xx'} = Cov(\mu_x, \mu_{x'})$. We show how to handle correlated beliefs below.

Parametric linear models We may derive a series of features $\phi_f(x)$, for $f \in \mathcal{F}$. Assume that we represent our belief using

$$f(x|\theta) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(x),$$

where $f(x|\theta) \approx \mathbb{E}F(x, W)$ is our estimate of $\mathbb{E}F(x, W)$. We now treat θ as the unknown parameter, where we might assume that the vector θ is described by a multivariate normal distribution $N(\theta^0, \Sigma^{\theta,0})$, although coming up with these priors (in the parameter space) can be tricky.

Parametric nonlinear models Our belief model might be nonlinear in θ. For example, we might use a logistic regression

$$f(x|\theta) \quad = \quad \frac{e^{U(x|\theta)}}{1 + e^{U(x|\theta)}}, \tag{7.37}$$

where $U(x|\theta)$ is a linear model given by

$$U(x|\theta) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_M.$$

As we show below, belief models that are nonlinear in the parameters can cause some difficulty, but we can circumvent this by using a sampled belief model, where we assume the uncertain θ is one of the set $\{\theta_1, \ldots, \theta_K\}$. Let p_k^n be the probability that $\theta = \theta_k$, which means that $p^n = (p_k^n),\ k = 1, \ldots, K$ is our belief at time n.

Nonparametric models Simpler nonparametric models are primarily local approximations, so we could use constant, linear or nonlinear models defined over local regions. More advanced models include neural networks (the kind known as "deep learners") or support vector machines, both of which were introduced in chapter 3.

Below we show how to calculate the knowledge gradient for each of these belief models, with the exception of the nonparametric models (listed for completeness).

7.7.2 The knowledge gradient for offline (terminal reward) learning

The knowledge gradient seeks to learn about the value of different actions by maximizing the value of information from a single observation. Let S^n be our belief state about the value of each action a. The knowledge gradient uses a Bayesian model, so

$$S^n = (\bar{\mu}_x^n, \sigma_x^{2,n})_{x \in \mathcal{X}},$$

captures the mean and variance of our belief about the true value $\mu_x = \mathbb{E}F(x, W)$, where we also assume that $\mu_x \sim N(\bar{\mu}_x^n, \sigma_x^{2,n})$.

The value of being in belief state S^n is given by

$$V^n(S^n) = \mu_{x^n},$$

where x^n is the choice that appears to be best given what we know after n experiments, calculated using

$$x^n = \arg\max_{x' \in \mathcal{X}} \bar{\mu}_{x'}^n.$$

If we choose action x^n, we then observe $W_{x^n}^{n+1}$ which we then use to update our estimate of our belief about μ_x using our Bayesian updating equations that we gave in section 7.2.

The value of state $S^{n+1}(x)$ when we try action x is given by

$$V^{n+1}(S^{n+1}(x)) = \max_{x' \in \mathcal{X}} \bar{\mu}_{x'}^{n+1}(x).$$

where $\bar{\mu}_{x'}^{n+1}(x)$ is the updated estimate of $\mathbb{E}\mu$ given S^n (that is, our estimate of the distribution of μ after n experiments), and the result of implementing x and observing W_x^{n+1}. We have to decide which experiment to run after the nth observation, so we have to work with the expected value of running experiment x, given by

$$\mathbb{E}\{V^{n+1}(S^{n+1}(x))|S^n\} = \mathbb{E}\{\max_{x' \in \mathcal{X}} \bar{\mu}_{x'}^{n+1}(x)|S^n\}.$$

The knowledge gradient is then given by

$$\nu_x^{KG,n} = \mathbb{E}\{V^{n+1}(S^M(S^n, x, W^{n+1}))|S^n, x\} - V^n(S^n),$$

which is equivalent to

$$\nu^{KG}(x) = \mathbb{E}\{\max_{x'} \bar{\mu}^{n+1}_{x'}(x)|S^n\} - \max_{x'} \bar{\mu}^n_{x'}. \tag{7.38}$$

Here, $\bar{\mu}^{n+1}(x)$ is the updated value of $\bar{\mu}^n$ after running an experiment with setting $x = x^n$, after which we observe W^{n+1}_x. Since we have not yet run the experiment, W^{n+1}_x is a random variable, which means that $\bar{\mu}^{n+1}(x)$ is random. In fact, $\bar{\mu}^{n+1}(x)$ is random for two reasons. To see this, we note that when we run experiment x, we observe an updated value from

$$W^{n+1}_x = \mu_x + \varepsilon^{n+1}_x,$$

where $\mu_x = \mathbb{E}F(x, W)$ is the true value, while ε^{n+1}_x is the noise in the observation. This introduces two forms of uncertainty: the unknown truth μ_x, and the noise ε^{n+1}_x. Thus, it would be more accurate to write equation (7.38) as

$$\nu^{KG}(x) = \mathbb{E}_\mu\{\mathbb{E}_{W|\mu} \max_{x'} \bar{\mu}^{n+1}_{x'}(x)|S^n\} - \max_{x'} \bar{\mu}^n_{x'}. \tag{7.39}$$

where the first expectation $\mathbb{E}_\mu$ is conditioned on our belief state S^n, while the second expectation $\mathbb{E}_{W|\mu}$ is over the experimental noise W given our distribution of belief about the truth μ.

To illustrate how equation (7.39) is calculated, imagine that μ takes on values $\{\mu_1, \ldots, \mu_K\}$, and that p^μ_k is the probability that $\mu = \mu_k$. Assume that μ is the mean of a Poisson distribution describing the number of customers W that click on a website and assume that

$$P^W[W = \ell|\mu = \mu_k] = \frac{\mu_k^\ell e^{-\mu_k}}{\ell!}.$$

We would then compute the expectation in equation (7.39) using

$$\nu^{KG}(x) = \sum_{k=1}^{K} \left(\sum_{\ell=0}^{\infty} \left(\max_{x'} \bar{\mu}^{n+1}_{x'}(x|W = \ell) \right) P^W[W = \ell|\mu = \mu_k] \right) p^\mu_k - \max_{x'} \bar{\mu}^n_{x'}.$$

where $\bar{\mu}^{n+1}_{x'}(x|W = \ell)$ is the updated estimate of $\bar{\mu}^n_{x'}$ if we run experiment x (which might be a price or design of a website) and we then observe $W = \ell$. The updating would be done using any of the recursive updating equations described in chapter 3.

We now want to capture how well we can solve our optimization problem, which means solving $\max_{x'} \bar{\mu}^{n+1}_{x'}(x)$. Since $\bar{\mu}^{n+1}_{x'}(x)$ is random (since we have to pick x before we know W^{n+1}), then $\max_{x'} \bar{\mu}^{n+1}_{x'}(x)$ is random. This is why we have to take the expectation, which is conditioned on S^n which captures what we know now.

Computing a knowledge gradient policy for independent beliefs is extremely easy. We assume that all rewards are normally distributed, and that we start with an initial estimate of the mean and variance of the value of decision x, given by

$\bar{\mu}^0_x$ = The initial estimate of the expected reward from making decision x,
$\bar{\sigma}^0_x$ = The initial estimate of the standard deviation of our belief about μ.

Each time we make a decision we receive a reward given by

$$W^{n+1}_x = \mu_x + \varepsilon^{n+1},$$

where μ_x is the true expected reward from action x (which is unknown) and ε is the experimental error with standard deviation σ_W (which we assume is known).

The estimates $(\bar{\mu}_x^n, \bar{\sigma}_x^{2,n})$ are the mean and variance of our belief about μ_x after n observations. We are going to find that it is more convenient to use the idea of *precision* (as we did in chapter 3) which is the inverse of the variance. So, we define the precision of our belief and the precision of the experimental noise as

$$\begin{aligned}\beta_x^n &= 1/\bar{\sigma}_x^{2,n},\\ \beta^W &= 1/\sigma_W^2.\end{aligned}$$

If we take action x and observe a reward W_x^{n+1}, we can use Bayesian updating to obtain new estimates of the mean and variance for action x, following the steps we first introduced in section 3.4. To illustrate, imagine that we try an action x where $\beta_x^n = 1/(20^2) = 0.0025$, and $\beta^W = 1/(40^2) = .000625$. Assume $\bar{\mu}_x^n = 200$ and that we observe $W_x^{n+1} = 250$. The updated mean and precision are given by

$$\begin{aligned}\bar{\mu}_x^{n+1} &= \frac{\beta_x^n \bar{\mu}_x^n + \beta^W W_x^{n+1}}{\beta_x^n + \beta^W}\\ &= \frac{(.0025)(200) + (.000625)(250)}{.0025 + .000625}\\ &= 210.\\ \beta_x^{n+1} &= \beta_x^n + \beta^W\\ &= .0025 + .000625\\ &= .003125.\end{aligned}$$

We next find the variance of the *change* in our estimate of μ_x assuming we choose to sample action x in iteration n. For this we define

$$\begin{aligned}\tilde{\sigma}_x^{2,n} &= \mathit{Var}[\bar{\mu}_x^{n+1} - \bar{\mu}_x^n | S^n] \qquad (7.40)\\ &= \mathit{Var}[\bar{\mu}_x^{n+1} | S^n]. \qquad (7.41)\end{aligned}$$

We use the form of equation (7.40) to highlight the definition of $\tilde{\sigma}_x^{2,n}$ as the change in the variance given what we know at time n, but when we condition on what we know (captured by S^n) it means that $\mathit{Var}[\bar{\mu}_x^n | S^n] = 0$ since $\bar{\mu}_x^n$ is just a number at time n.

With a little work, we can write $\tilde{\sigma}_x^{2,n}$ in different ways, including

$$\begin{aligned}\tilde{\sigma}_x^{2,n} &= \bar{\sigma}_x^{2,n} - \bar{\sigma}_x^{2,n+1}, \qquad (7.42)\\ &= \frac{(\bar{\sigma}_x^{2,n})}{1 + \sigma_W^2/\bar{\sigma}_x^{2,n}}. \qquad (7.43)\end{aligned}$$

Equation (7.42) expresses the (perhaps unexpected) result that $\tilde{\sigma}_x^{2,n}$ measures the change in the estimate of the standard deviation of the reward from decision x from iteration $n-1$ to n. Using our numerical example, equations (7.42) and (7.43) both produce the result

$$\begin{aligned}\tilde{\sigma}_x^{2,n} &= 400 - 320 = 80\\ &= \frac{40^2}{1 + \frac{10^2}{40^2}} = 80.\end{aligned}$$

Finally, we compute

$$\zeta_x^n = -\left| \frac{\bar{\mu}_x^n - \max_{x' \neq x} \bar{\mu}_{x'}^n}{\tilde{\sigma}_x^n} \right|.$$

ζ_x^n is called the *normalized influence* of decision x. It gives the number of standard deviations from the current estimate of the value of decision x, given by $\bar{\mu}_x^n$, and the best alternative other than decision x. We then find

$$f(\zeta) = \zeta\Phi(\zeta) + \phi(\zeta),$$

where $\Phi(\zeta)$ and $\phi(\zeta)$ are, respectively, the cumulative standard normal distribution and the standard normal density. Thus, if Z is normally distributed with mean 0, variance 1, $\Phi(\zeta) = \mathbb{P}[Z \leq \zeta]$ while

$$\phi(\zeta) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{\zeta^2}{2}\right).$$

The knowledge gradient algorithm chooses the decision x with the largest value of $\nu_x^{KG,n}$ given by

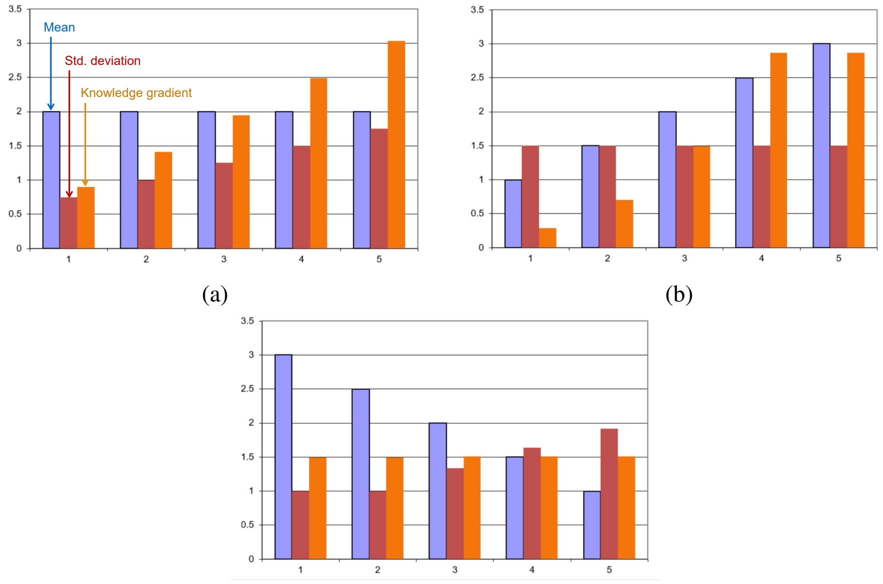
$$\nu_x^{KG,n} = \tilde{\sigma}_x^n f(\zeta_x^n).$$

The knowledge gradient algorithm is quite simple to implement. Table 7.4 illustrates a set of calculations for a problem with five options. $\bar{\mu}$ represents the current estimate of the value of each action, while $\bar{\sigma}$ is the current standard deviation of μ. Options 1, 2 and 3 have the same value for $\bar{\sigma}$, but with increasing values of $\bar{\mu}$. The table illustrates that when the variance is the same, the knowledge gradient prefers the decisions that appear to be the best. Decisions 3 and 4 have the same value of $\bar{\mu}$, but decreasing values of $\bar{\sigma}$, illustrating that the knowledge gradient prefers decisions with the highest variance. Finally, decision 5 appears to be the best of all the decisions, but has the lowest variance (meaning that we have the highest confidence in this decision). The knowledge gradient is the smallest for this decision out of all of them.

Decision	$\bar{\mu}$	$\bar{\sigma}$	$\tilde{\sigma}$	ζ	$f(z)$	KG index
1	1.0	2.5	1.569	-1.275	0.048	0.075
2	1.5	2.5	1.569	-0.956	0.090	0.142
3	2.0	2.5	1.569	-0.637	0.159	0.249
4	2.0	2.0	1.400	-0.714	0.139	0.195
5	3.0	1.0	0.981	-1.020	0.080	0.079

Table 7.4 The calculations behind the knowledge gradient algorithm

The knowledge gradient trades off between how well an alternative is expected to perform, and how uncertain we are about this estimate. Figure 7.4 illustrates this tradeoff. Figure 7.4(a) shows five alternatives, where the estimates are the same across all three alternatives, but with increasing standard deviations. Holding the mean constant, the knowledge gradient increases with standard deviation of the estimate of the mean. Figure 7.4(b) repeats this exercise, but now holding the standard deviation the same, with increasing means, showing that the knowledge gradient increases with the estimate of the mean. Finally, figure 7.4(c) varies the estimates of the mean and standard deviation so that the knowledge gradient stays constant, illustrating the tradeoff between the estimated mean and its uncertainty.



(a) (b)

(c)

Figure 7.4 The knowledge gradient for lookup table with independent beliefs with equal means (a), equal variances (b), and adjusting means and variances so that the KG is equal (c).

This tradeoff between the expected performance of a design, and the uncertainty about its performance, is a feature that runs through well designed policies. However, all the other policies with this property (interval estimation, upper confidence bounding, Gittins indices), achieve this with indices that consist of the sum of the expected performance and a term that reflects the uncertainty of an alternative. The knowledge gradient, however, achieves this behavior without constructing the policy in this way. In fact, it is only when we extend the knowledge gradient to online settings where we learn in the field (in section 7.7.4) do we obtain an index that features the sum of the expected value of an alternative and a term that captures the value of information (from the offline knowledge gradient).

7.7.3 Concavity of information

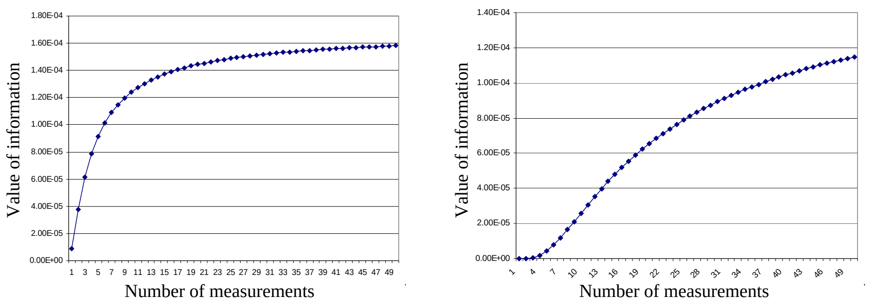
We pause before showing how to compute the knowledge gradient for online (cumulative reward) problems by addressing an important issue that arises when dealing with the marginal value of information.

Imagine that instead of doing a single experiment, that we can repeat our evaluation of alternative x n_x times. If we are using a lookup table belief model, this means the updated precision is

$$\beta_x^{n+1}(n_x) = \beta_x^n + n_x \beta^W,$$

where as before, β^W is the precision of a single experiment. If we repeat this experiment n_x times, this is the same as doing one experiment with precision $n_x \beta^W$.

Now consider the value of information from doing this repeated experiment. The value of the n_x experiments is the same as the knowledge gradient for a single experiment with



7.5(a) 7.5(b)

Figure 7.5 Value of making n observations. In (a), the value of information is concave, while in (b) the value of information follows an S-curve.

precision $n_x\beta^W$, which we write $\nu_x^{KG,n}(n_x)$. This raises the question: what does the function $\nu_x^{KG,n}(n_x)$ look like?

It is tempting to claim that $\nu_x^{KG,n}(n_x)$ is concave in n_x, since each experiment might be reasonably expected to contribute less, on the margin. You might be right, as show in figure 7.5(a). However, it is also possible that the value of information looks like an S-curve, as shown in 7.5(b). What explains the difference?

The S-curve behavior in figure 7.5(b) arises when experiments are noisy, which means that a single experiment contributes little information. This behavior is actually quite common, especially when the outcome of an experiment is a success or failure (perhaps indicated by 1 or 0). It is very easy to check for this behavior. All that is required is to compute $\nu_x^{KG,n}(n_x)$ for $n_x = 1$ and $n_x = 2$. If $\nu_x^{KG,n}(2) - \nu_x^{KG,n}(1) > \nu_x^{KG,n}(1)$, which means the marginal value of information is initially increasing, then the value of information is non-concave. When this is the case, $\nu_x^{KG,n}(1)$, which is the standard knowledge gradient, may provide very little information (in fact, it may be virtually zero).

There are several ways to handle the situation. One is to use the KG(*) algorithm, where we find the value of n_x that produces the highest *average* value of information. We first compute n_x^* from

$$n_x^* = \arg\max_{n_x > 0} \frac{v_x(n_x)}{n_x}. \tag{7.44}$$

This is illustrated in figure 7.6. We do this for each x, and then run the experiment with the highest value of $\frac{v_x(n_x)}{n_x}$. Note that we are not requiring that each experiment be repeated n_x^* times; we only use this to produce a new index (the maximum average value of information), which we use to identify the next single experiment.

A more general policy uses the concept of *posterior reshaping*. The idea is quite simple. Introduce a repetition parameter η where we let the precision of an experiment be given by

$$\beta_x^{n+1}(\eta) = \beta_x^n + \eta\beta^W.$$

Now let $\nu_x^{KG,n}(\eta)$ be the knowledge gradient when we use repetition factor η. Our knowledge gradient policy would be still given by

$$X^{KG}(S^n|\eta) = \arg\max_x \nu_x^{KG,n}(\eta).$$

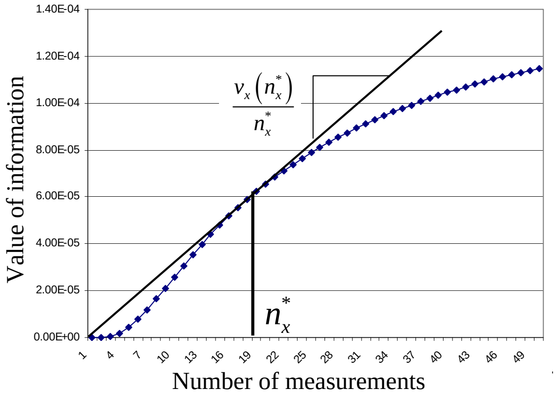


Figure 7.6 The KG(*) policy, which maximizes the average value of a series of experiments testing a single alternative.

We now have a tunable parameter, but this is the price of managing this complexity. The value of using a tunable parameter is that the tuning process implicitly captures the effect of the experimental budget N.

7.7.4 The knowledge gradient for online (cumulative reward) learning

We have derived the knowledge gradient for offline (finel reward) learning problems. Our choice of action does not depend directly on how good the action is; we only care about how much information we gain from trying an action to improve the quality of the final decision.

There are many online applications of dynamic programming where there is an operational system which we would like to optimize in the field. In these settings, we have to live with the rewards from each experiment. As a result, we have to strike a balance between the value of an action and the information we gain that may improve our choice of actions in the future. This is precisely the tradeoff that is made by Gittins indices for the multiarmed bandit problem.

It turns out that the knowledge gradient is easily adapted for online problems. As before, let $\nu_x^{KG,n}$ be the offline knowledge gradient, giving the value of observing action x, measured in terms of the improvement in a single decision. Now imagine that we have a budget of N decisions. After having made n decisions (which means, n observations of the value of different actions), if we observe $x = x^n$ which allows us to observe W_x^{n+1}, we received an expected reward of $\mathbb{E}^n W_x^{n+1} = \bar{\mu}_x^n$, and obtain information that improves the contribution from a single decision by $\nu_x^{KG,n}$. However, we have $N - n$ more decisions to make. Assume that we learn from the observation of W_x^{n+1} by choosing $x^n = x$, but we do not allow ourselves to learn anything from future decisions. This means that the remaining $N - n$ decisions have access to the same information.

From this analysis, the knowledge gradient for online applications consists of the expected value of the single-period contribution of the experiment, plus the improvement in all the remaining decisions in our horizon. This implies

$$\nu_x^{OLKG,n} = \bar{\mu}_x^n + (N-n)\nu_x^{KG,n}. \tag{7.45}$$

If we have an infinite horizon problem with discount factor γ,

$$\nu_x^{OLKG,n} = \bar{\mu}_x^n + \frac{\gamma}{1-\gamma}\nu_x^{KG,n}. \tag{7.46}$$

We note that the logic for incorporating correlated beliefs for the offline knowledge gradient can now be directly applied to online problems.

7.7.5 Characteristics and properties of the knowledge gradient

Some characteristics of the knowledge gradient are

1) The knowledge gradient is Bayesian, which means it can work with prior knowledge from a domain expert.

2) It can handle a variety of belief models (lookup table with independent or correlated beliefs, linear or nonlinear parametric models, nonparametric models).

3) The knowledge gradient is typically harder to compute, and sometimes much harder (depending on the belief model) than policies such as UCB, IE or Thompson sampling, although it is much easier than the Gittins index policy which is limited to lookup table belief models. Its relative computational complexity, combined with its ability to incorporate prior knowledge, means that it is best suited to problems where function evaluations are expensive (for example, they may require complex simulations or field experiments).

The knowledge gradient for offline (terminal reward) problems enjoys several mathematical properties, including

- Property 1: The knowledge gradient is always positive, $\nu_x^{KG,n} \geq 0$ for all x. Thus, if the knowledge gradient of an alternative is zero, that means we will not run another experiment with those settings.

- Property 2: The knowledge gradient policy is optimal (by construction) if we are going to make exactly one experiment.

- Property 3: If there are only two choices, the knowledge gradient policy is optimal for any experimental budget N.

- Property 4: If N is our experimental budget, the knowledge gradient policy is guaranteed to find the best alternative as N is allowed to be big enough. That is, if x^N is the solution we obtain after N experiments, and

  $$x^* = \arg\max \mu_x$$

  is the true best alternative, then $x^N \rightarrow x^*$ as $N \rightarrow \infty$. This property is known as asymptotic optimality.

- Property 5: There are many heuristic policies that are asymptotically optimal (for example, pure exploration, mixed exploration-exploitation, epsilon-greedy exploration and Boltzmann exploration). But none of these heuristic policies is myopically optimal. The knowledge gradient policy is the only pure policy (an alternative term would be to say it is the only stationary policy) that is both myopically and asymptotically optimal.

- Property 6: The knowledge gradient has no tunable algorithmic parameters. Heuristics such as the Boltzmann policy (θ^{Boltz} in equation (7.18)) and interval estimation (θ^{IE} in equation (7.17)) have tunable algorithmic parameters. The knowledge gradient has no such parameters (although these can be introduced, as we do below), but as with all Bayesian methods, it does require a prior which allows us to take advantage of domain knowledge (but then it requires that we have access to domain knowledge).

There is one more very nice property enjoyed by the knowledge gradient, which is

- Property 7: The knowledge gradient can be adapted for either final-reward or cumulative-reward settings.

This deserves some discussion. Other policies in the class of cost function approximations such as interval estimation and upper confidence bounding that exhibit tunable parameters can be tuned for either final-reward or cumulative-reward settings. The knowledge gradient does not, in principle, have to be tuned, although this is primarily for problems where the value of information is concave. If the value of information is non-concave, then we recommend tuning.

The knowledge gradient is not an optimal policy for collecting information, but it has been found to work quite well in numerous experimental research (as long as care is used when the value of information is nonconcave). We repeat that the knowledge gradient is much more difficult to compute than other policies such as UCB policies, interval estimation and Thompson sampling, which makes it well suited to problems where experiments are expensive (and take time).

7.8 THE KNOWLEDGE GRADIENT FOR MORE GENERAL BELIEF MODELS

The feature that most distinguishes the knowledge gradient from tunable policies such as interval estimation, upper confidence bounding and Boltzmann exploration is that the belief model is built into the policy. This is because the knowledge gradient is in the class of direct lookahead policies which require that we solve a model in the future, which requires that we capture the updating of the belief model in the policy. This leads to high quality solutions, but the complexity and computational cost goes up accordingly.

7.8.1 Knowledge gradient for correlated beliefs

A particularly important feature of the knowledge gradient is that it can be adapted to handle the important problem of correlated beliefs. In fact, the vast majority of real applications exhibit some form of correlated beliefs. Some examples are given below.

■ **EXAMPLE 7.1**

Correlated beliefs can arise when we are maximizing a continuous surface (nearby points will be correlated) or choosing subsets (such as the location of a set of facilities) which produce correlations when subsets share common elements. If we are trying to estimate a continuous function, we might assume that the covariance matrix satisfies

$$Cov(x, x') \propto e^{-\rho\|x-x'\|},$$

where ρ captures the relationship between neighboring points. If x is a vector of $0's$ and $1's$ indicating elements in a subset, the covariance might be proportional to the number of 1's that are in common between two choices.

■ **EXAMPLE 7.2**

Netflix needs to test different movies to see which are most popular for a particular account. Each movie can be described by a set of attributes (such as genre, actors, setting). By selecting one movie, we can learn about movies with similar attributes.

■ **EXAMPLE 7.3**

There are about two dozen drugs for reducing blood sugar, divided among four major classes. Trying a drug in one class can provide an indication of how a patient will respond to other drugs in that class.

■ **EXAMPLE 7.4**

A materials scientist is testing different catalysts in a process to design a material with maximum conductivity. Prior to running any experiment, the scientist is able to estimate the likely relationship in the performance of different catalysts, shown in table 7.5. The catalysts that share an Fe (iron) or Ni (nickel) molecule show higher correlations.

	1.4nmFe	1nmFe	2nmFe	10nm-Fe	2nmNi	Ni0.6nm	10nm-Ni
1.4nmFe	1.0	0.7	0.7	0.6	0.4	0.4	0.2
1nmFe	0.7	1.0	0.7	0.6	0.4	0.4	0.2
2nmFe	0.7	0.7	1.0	0.6	0.4	0.4	0.2
10nmFe	0.6	0.6	0.6	1.0	0.4	0.3	0.0
2nmNi	0.4	0.4	0.4	0.4	1.0	0.7	0.6
Ni0.6nm	0.4	0.4	0.4	0.3	0.7	1.0	0.6
10nmNi	0.2	0.2	0.2	0.0	0.6	0.6	1.0

Table 7.5 Correlation matrix describing the relationship between estimated performance of different catalysts, as estimated by an expert.

Constructing the covariance matrix involves incorporating the structure of the problem. This may be relatively easy, as with the covariance between discretized choices of a continuous surface.

There is a more compact way of updating our estimate of $\bar{\mu}^n$ in the presence of correlated beliefs. Let $\lambda^W = \sigma^2_W = 1/\beta^W$ (this is basically a trick to get rid of that nasty square). Let $\Sigma^{n+1}(x)$ be the updated covariance matrix given that we have chosen to evaluate alternative x, and let $\tilde{\Sigma}^n(x)$ be the change in the covariance matrix due to evaluating x, which is given by

$$\begin{aligned}\tilde{\Sigma}^n(x) &= \Sigma^n - \Sigma^{n+1}, \\ &= \frac{\Sigma^n e_x (e_x)^T \Sigma^n}{\Sigma^n_{xx} + \lambda^W}.\end{aligned}$$

Now define the vector $\tilde{\sigma}^n(x)$, which gives the square root of the change in the variance due to measuring x, which is given by

$$\tilde{\sigma}^n(x) = \frac{\Sigma^n e_x}{\sqrt{\Sigma^n_{xx} + \lambda^W}}. \tag{7.47}$$

Let $\tilde{\sigma}_i(\Sigma, x)$ be the component $(e_i)^T \tilde{\sigma}(x)$ of the vector $\tilde{\sigma}(x)$, and let $Var^n(\cdot)$ be the variance given what we know after n experiments. We note that if we evaluate alternative x^n, then

$$\begin{aligned}Var^n\left[W^{n+1} - \bar{\mu}^n_{x^n}\right] &= Var^n\left[\mu_{x^n} + \varepsilon^{n+1}\right] \\ &= \Sigma^n_{x^n x^n} + \lambda^W. \qquad (7.48)\end{aligned}$$

Next define the random variable

$$Z^{n+1} = (W^{n+1} - \bar{\mu}^n_{x^n}) / \sqrt{Var^n\left[W^{n+1} - \bar{\mu}^n_{x^n}\right]}.$$

We can now rewrite (7.29) for updating our beliefs about the mean as

$$\bar{\mu}^{n+1} = \bar{\mu}^n + \tilde{\sigma}(x^n) Z^{n+1}. \tag{7.49}$$

Note that $\bar{\mu}^{n+1}$ and $\bar{\mu}^n$ are vectors giving beliefs for all alternatives, not just the alternative x^n that we tested. The knowledge gradient policy for correlated beliefs is computed using

$$\begin{aligned}X^{KG}(s) &= \arg\max_x \mathbb{E}\left[\max_i \mu_i^{n+1} \mid S^n = s\right] \qquad (7.50) \\ &= \arg\max_x \mathbb{E}\left[\max_i \left(\bar{\mu}^n_i + \tilde{\sigma}_i(x^n) Z^{n+1}\right) \mid S^n, x\right].\end{aligned}$$

where Z is a scalar, standard normal random variable. The problem with this expression is that the expectation is harder to compute, but a simple algorithm can be used to compute the expectation exactly. We start by defining

$$h(\bar{\mu}^n, \tilde{\sigma}(x)) = \mathbb{E}\left[\max_i \left(\bar{\mu}^n_i + \tilde{\sigma}_i(x^n) Z^{n+1}\right) \mid S^n, x = x^n\right]. \tag{7.51}$$

Substituting (7.51) into (7.50) gives us

$$X^{KG}(s) = \arg\max_x h(\bar{\mu}^n, \tilde{\sigma}(x)). \tag{7.52}$$

Let $a_i = \bar{\mu}^n_i$, $b_i = \tilde{\sigma}_i(\Sigma^n, x^n)$, and let Z be our standard normal deviate. Now define the function $h(a, b)$ as

$$h(a, b) = \mathbb{E} \max_i \left(a_i + b_i Z\right). \tag{7.53}$$

Both a and b are M-dimensional vectors. Sort the elements b_i so that $b_1 \leq b_2 \leq \ldots$ so that we get a sequence of lines with increasing slopes, as depicted in figure 7.7. There are

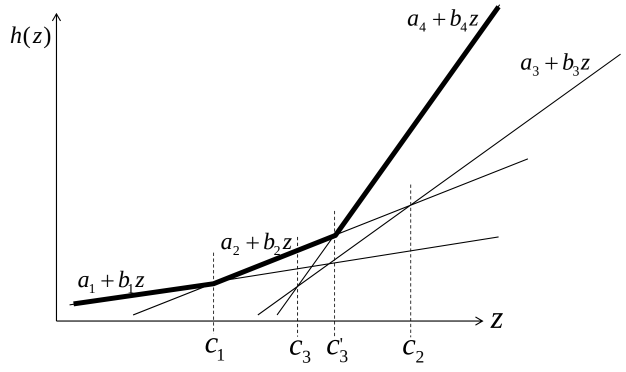


Figure 7.7 Regions of z over which different choices dominate. Choice 3 is always dominated.

ranges for z over a particular line may dominate the other lines, and some lines may be dominated all the time (such as alternative 3).

We need to identify and eliminate the dominated alternatives. To do this we start by finding the points where the lines intersect. The lines $a_i + b_i z$ and $a_{i+1} + b_{i+1} z$ intersect at

$$z = c_i = \frac{a_i - a_{i+1}}{b_{i+1} - b_i}.$$

For the moment, we are going to assume that $b_{i+1} > b_i$. If $c_{i-1} < c_i < c_{i+1}$, then we can find a range for z over which a particular choice dominates, as depicted in figure 7.7. A line is dominated when $c_{i+1} < c_i$, at which point they are dropped from the set. Once the sequence c_i has been found, we can compute (7.50) using

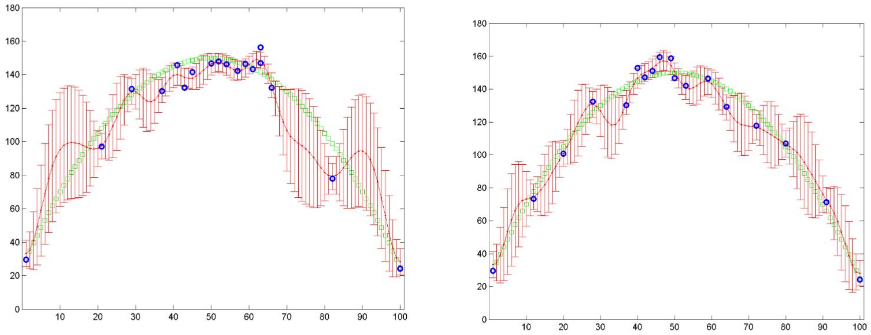
$$h(a, b) = \sum_{i=1}^{M} (b_{i+1} - b_i) f(-|c_i|),$$

where as before, $f(z) = z\Phi(z) + \phi(z)$. Of course, the summation has to be adjusted to skip any choices i that were found to be dominated.

It is important to recognize that there is more to incorporating correlated beliefs than simply using the covariances when we update our beliefs after an experiment. With this procedure, we anticipate the updating before we even perform an experiment.

The ability to handle correlated beliefs in the choice of what experiment to perform is an important feature that has been overlooked in other procedures. It makes it possible to make sensible choices when our experimental budget is much smaller than the number of potential choices we have to evaluate. There are, of course, computational implications. It is relatively easy to handle dozens or hundreds of alternatives, but as a result of the matrix calculations, it becomes expensive to handle problems where the number of potential choices is in the thousands. If this is the case, it is likely the problem has special structure. For example, we might be discretizing a p-dimensional parameter surface. If this is the case, it may make sense to consider the adaptation of the knowledge gradient for problems where the belief structure can be represented using a parametric model.

A reasonable question to ask is: given that the correlated KG is considerably more complex than the knowledge gradient policy with independent beliefs, what is the value of



(a)

(b)

Figure 7.8 (a) Sampling pattern from knowledge gradient using independent beliefs; (b) sampling pattern from knowledge gradient using correlated beliefs.

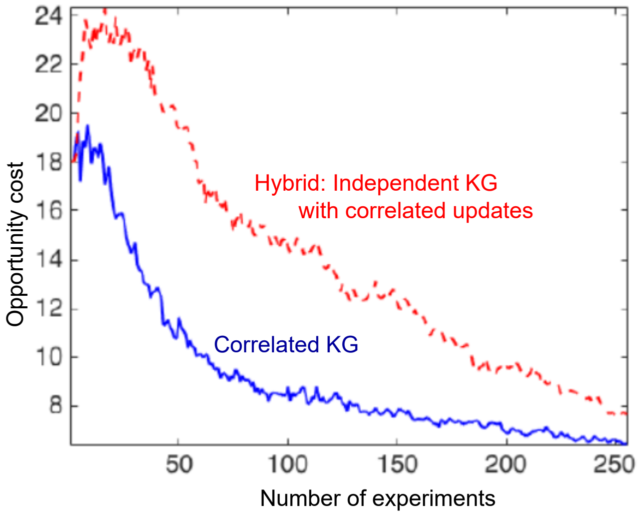


Figure 7.9 Comparison of correlated KG policy against a KG policy with independent beliefs, but using correlated updates, showing the improvement when using the correlated KG policy.

using correlated KG? Figure 7.8(a) shows the sampling pattern when learning a quadratic function, starting with a uniform prior, when using the knowledge gradient with independent beliefs for the learning policy, but using correlated beliefs to update beliefs after an experiment has been run. This policy tends to produce sampling that is more clustered in the region near the optimum. Figure 7.8(b) shows the sampling pattern for the knowledge gradient policy with correlated beliefs, showing a more uniform pattern that shows a better spread of experiments.

So, the correlated KG logic seems to do a better job of exploring, but how well does it work? Figure 7.9 shows the opportunity cost for each policy, where smaller is better. For this example, the correlated KG works quite a bit better, probably due to the tendency of the correlated KG policy to do explore more efficiently.

While these experiments suggest strong support for the correlated KG policy when we have correlated beliefs, we need to also note that tunable CFA-based policies such as interval estimation or the UCB policies can also be tuned in the context of problems with correlated beliefs. The tradeoff is that the correlated KG policy does not require tuning, but is much more difficult to implement. A tuned CFA policy requires tuning (which can be a challenge) but is otherwise trivial to implement.

7.8.2 Linear belief models

While lookup table belief models will always find applications, they quickly become cumbersome when there are many alternatives x, as will almost always happen when x is a vector. For this reason, the field of statistics has often turned to linear models of the form

$$\mathbb{E}F(x,W) \quad \approx \quad f(x|\theta) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(x).$$

With a lookup table, each estimate $\bar{\mu}_x^n$ was a different parameter (one for each value of x). With a linear model, we have only to estimate the vector θ, which is dimensioned by the number of features $\mathcal{F}$. It turns out that rather than working with a covariance matrix Σ that is dimensioned by the number of possible decisions x (which easily stretches to millions when x is a vector), we can work with a matrix Σ^θ that is dimensioned by the number of features in $\mathcal{F}$ with an element $\Sigma_{ff'} = Cov(\theta_f, \theta_{f'})$.

When we use a linear belief model (but still have discrete alternatives x), we are basically creating a problem with correlated beliefs, where the correlation between $\mu_x = \mathbb{E}F(x,W)$ and $\mu_{x'} = \mathbb{E}F(x',W)$ arises because of the structure of the linear belief model. For this reason, we are going to build on the tools we developed in section **??** for correlated beliefs. There, we introduced the function

$$h(a,b) = \mathbb{E} \max_i \big(a_i + b_i Z\big),$$

where $a_i = \bar{\mu}_i^n$, $b_i(j) = \tilde{\sigma}_i(j)$, and Z is a standard normal deviate. Here, the index j corresponds to the experiment $x = x_j$ we are thinking of performing, and the index i corresponds to the design parameters $x' = x_i$ that we might choose given our current belief. For the remainder of this section, we continue to use (i,j) to be consistent with the presentation of correlated beliefs using lookup tables.

Using our linear belief model $f(i|\theta)$, after n experiments, we would have an estimate of our parameter $\theta = \theta^n$ which means that we would compute $a_i = f(x' = x_i|\theta^n)$. Deferring for a moment the calculation of $b_i(j) = \tilde{\sigma}_i(j)$ (this is the hard part), we still calculate $h(a,b)$ using

$$h(a,b(j)) = \sum_{i=1}^{M-1} (b_{i+1}(j) - b_i(j)) f(-|c_i(j)|) \tag{7.54}$$

where

$$c_i(j) = \frac{a_i - a_{i+1}}{b_{i+1}(j) - b_i(j)}$$

is the point where the lines $a_i + b_i z$ and $a_j + b_j z$ intersect.

The major computational challenge arises when computing $\tilde{\sigma}^n(j)$, which is a vector giving the change in the variance of each alternative i if we choose to evaluate alternative j.

Keep in mind that unlike the lookup table representation in section **??** where the number of alternatives might be in the hundreds (up to a thousand), we are now considering problems where the number of alternatives might be in the thousands or tens of thousands, which prevents us from creating a matrix Σ dimensioned by the number of alternatives. We are going to show that we can compute $\tilde{\sigma}^n(j)$ using the matrix Σ^θ, which is much smaller than Σ.

From our presentation of the knowledge gradient for correlated beliefs, we found that

$$\tilde{\Sigma}^n(j) \quad = \quad \frac{\Sigma^n e_j (e_j)^T \Sigma^n}{\Sigma^n_{jj} + \sigma^2_W},$$

which gives us the change in the covariance matrix from measuring an alternative j. The matrix $\tilde{\Sigma}^n(j)$ is $M \times M$, which is quite large, but we only need the jth row, which we compute using

$$\tilde{\sigma}^n(j) \quad = \quad \frac{\Sigma^n e_j}{\sqrt{\Sigma^n_{jj} + \sigma^2_W}}. \tag{7.55}$$

Now is where we use the structure of our linear model. If there are a large number of choices, the matrix $\tilde{\Sigma}^n$ will be too expensive to compute. However, it is possible to work with the covariance matrix Σ^θ of the regression vector θ, which is dimensioned by the number of parameters in our regression function. To compute Σ^θ, we need to set up some variables. First let

$$x^n = \begin{pmatrix} x^n_1 \\ x^n_2 \\ \vdots \\ x^n_F \end{pmatrix}$$

be a vector of $F = |\mathcal{F}|$ independent variables corresponding to the nth observation. Next create the matrix X^n which consists of all of our experiments

$$X^n = \begin{pmatrix} x^1_1 & x^1_2 & & x^1_K \\ x^2_1 & x^2_2 & & x^2_F \\ \vdots & \vdots & \cdots & \vdots \\ x^n_1 & x^n_F & & x^n_F \end{pmatrix}.$$

We can use X^n to compute our regression vector by computing

$$\theta^n \quad = \quad [(X^n)^T X^n]^{-1}(X^n)^T Y^n.$$

We can use X^n to compute a compact form for the covariance matrix $\Sigma^{\theta,n}$ of θ. With a bit of algebra, it is possible to show that, after n observations, then

$$\begin{aligned} \Sigma^{\theta,n} &= [(X^n)^T X^n]^{-1}(X^n)^T X^n [(X^n)^T X^n]^{-1}\sigma^2_W \\ &= [(X^n)^T X^n]^{-1}\sigma^2_W. \end{aligned}$$

where σ^2_W is the variance of our observations. Of course, this seems to imply that we still have to invert $(X^n)^T X^n$. We can again use the tricks in section 3.8.1 to compute this inverse recursively.

Knowing that we can compute $\Sigma^{\theta,n}$ in an efficient way, we can quickly compute $\tilde{\sigma}^n$. Note that we do not need the entire matrix. We can show that

$$\Sigma^n = X\Sigma^{\theta,n}(X)^T.$$

Let X_j be the jth row of X. Then

$$\tilde{\sigma}^n(j) = X_j\Sigma^{\theta,n}X^T.$$

We still have to multiply the $K\times K$-dimensional matrix $\Sigma^{\theta,n}$ times the $K\times M$-dimensional matrix X^T, after which we have to compute equation (7.54) to find the knowledge gradient for each alternative. Even for problems with tens of thousands of alternatives, this can be executed in a few seconds. Now that we have an efficient way of computing $\tilde{\sigma}^n(j)$, we can apply the knowledge gradient for correlated beliefs described in section 7.8.1.

Thus, the knowledge gradient using a linear belief model is basically the same as the knowledge gradient for correlated beliefs using a lookup table model. The difference is that we use the linear model to create a short-cut that allows us to avoid enumerating the entire matrix Σ. Instead, the only matrix we need to explicitly manipulate is Σ^θ. We have used this to handle problems with up to 100,000 alternatives.

7.8.3 Nonlinear belief models

We have shown that we can calculate the knowledge gradient for linear models (lookup tables, or linear parametric models), but nonlinear models are another matter. Consider, for example, a problem with a binomial (success/failure) outcome W where the probability $W=1$ is given by a logistic regression

$$f^W(W=1|x,\theta) \quad = \quad \frac{e^{\sum_{f\in\mathcal{F}}\theta_f\phi_f(x)}}{1+e^{\sum_{f\in\mathcal{F}}\theta_f\phi_f(x)}}. \tag{7.56}$$

Now define the function

$$F(x,W) = W$$

where we would like to solve the maximization problem

$$\max_x \mathbb{E}F(x,W).$$

This problem arises in many settings. Some examples are:

■ **EXAMPLE 7.1**

The decision x might be the choice of movies to display on a website, where $W=1$ means that the customer chose a movie. This is one example of a *recommender system* where we want to make suggestions of products and services with the goal of maximizing the number of customers who accept the offer.

■ **EXAMPLE 7.2**

A physician might make a medical choice x with the goal of maximizing the likelihood of solving the patient's problem. In this setting, the vector x might be a mixture of patient attributes (which cannot be controlled) and medical decisions which can.

Given the probability model in equation (7.56), finding the optimal solution would be quite easy if we just knew θ since $\mathbb{E}F(x,W) = f^W(W=1|x,\theta)$. Rather, we have to learn θ through a sequence of decisions followed by observations of $F(x,W)$.

Now consider the knowledge gradient formula

$$\nu^{KG}(x) = \mathbb{E}_\theta \mathbb{E}_{W|\theta}\{\max_{x'} F(x', \theta^{n+1}(x))|S^n, x = x^n\} - \mathbb{E}_\theta\{\max_{x'} F(x', \theta)|S^n\}. \tag{7.57}$$

We note that the first expectation is over two sources of uncertainty, The first and most difficult is the uncertainty in θ, since θ is typically a vector, and potentially a high-dimensional one. The second is in the noisy outcome, which in this case requires sampling a binomial outcome from our logistic regression. The expectation in the second term is over the distribution of θ captured by our current belief in S^n. The problem is the expectation over the vector θ, which is imbedded inside a $\max$ operator.

Equation (7.57) could be computed when the function $F(x, \theta)$ was linear in θ, which provided simple closed-form formulas for $\theta^{n+1}(x)$, such as the Bayesian updating formulas (3.19)-(3.20) for the mean and precision when using a lookup table representation, or the recursive formulas for linear models given in section 3.8. However, when $F(x, \theta)$ is nonlinear in θ, as in equation (7.56), we lose these analytical updating formulas, which makes the calculation of the knowledge gradient in (7.57) intractable, primarily because of the expectation.

There is a way to overcome this problem. We are going to represent our uncertain true value of θ as taking on one of a finite (and not too large) set of discrete values $\theta_1, \ldots, \theta_K$. Let $p_k^n = \mathbb{P}[\theta = \theta^k]$, which is to say, the probability that $\theta = \theta_k$ after n experiments. The function $F(x, \theta)$ is approximated by

$$\overline{F}^n(x') \quad = \quad \sum_{k=1}^K p_k^n F(x', \theta_k).$$

Assume we run an experiment $x^n = x$, observe a random outcome W^{n+1} and use this to update our probabilities which we write as $p_k^{n+1}(x)$, which is a random variable before we have actually run the experiment. When we run an experiment $x^n = x$, we observe W^{n+1} with distribution

$$f^W(w|x, \theta) \quad = \quad \mathbb{P}[W^{n+1} = w|x, \theta],$$

which we will use shortly to compute the posterior distribution $p_k^{n+1}(x)$ of the probability that $\theta = \theta_k$ after observing W^{n+1}. We use the posterior probability to compute an updated estimate

$$\overline{F}^{n+1}(x'|x^n = x) \quad = \quad \sum_{k=1}^K p_k^{n+1}(x) F(x', \theta_k).$$

Using our assumption that θ takes on one of the outcomes in $(\theta_1, \ldots, \theta_K)$, we can write the knowledge gradient as

$$\nu^{KG}(x) = \mathbb{E}\left\{\max_{x'} \sum_{k=1}^K p_k^{n+1}(x) F(x', \theta_k)|S^n, x = x^n\right\} - \sum_{k=1}^K p_k^n F(x, \theta_k). \tag{7.58}$$

We have to resolve how to compute the posterior probability $p_k^{n+1}(x)$ and the expectation $\mathbb{E}$. Assume that our random outcome W from an experiment takes on one of the outcomes $w_1, \ldots, w_L$ with probability $\mathbb{P}[W = w_\ell] = p^W(w_\ell|x, \theta)$. If we observe $W = w_\ell$, then this enters the calculation of the posterior probability $p_k^{n+1}(x)$, which is calculated using Bayes theorem. Let H^n be the history

$$H^n = (S^0, x^0, W^1, x^1, \ldots, W^n).$$

The probability p_k^n can be stated as $\mathbb{P}[\theta = \theta_k|H^n]$. The posterior probability $p_k^{n+1}(x)$ is then calculated using

$$\begin{aligned}
p_k^{n+1}(x = x^n|H^{n+1}) &= \mathbb{P}[\theta = \theta_k|H^{n+1}] && (7.59)\\
&= \mathbb{P}[\theta = \theta_k|x^0, W^1, x^1, \ldots, x^n, W^{n+1}] && (7.60)\\
&\propto \mathbb{P}[W^{n+1}|x^n, \theta_k]\mathbb{P}[\theta = \theta_k|x^0, W^1, x^1, \ldots, x^n] && (7.61)\\
&= f^W(W^{n+1}|x^n, \theta_k)\mathbb{P}[\theta = \theta_k|H^n, x^n] && (7.62)\\
&= f^W(W^{n+1}|x^n, \theta_k)\mathbb{P}[\theta = \theta_k|H^n] && (7.63)\\
&= f^W(W^{n+1}|x^n, \theta_k)p_k^n. && (7.64)
\end{aligned}$$

Equation (7.59) is the definition of the posterior probability $p_k^{n+1}(x = x^n)$, which we write with the full history in (7.60). Equation (7.61) is from Bayes theorem, where we dropped the denominator (hence the use of $\propto$). Equation (7.62) rewrites the first probability using H^n, and then we drop x^n in (7.63) since the probability that $\theta = \theta_k$ depends only on H^n, not the final decision. We finish by simply substituting in p_k^n.

Using Bayes theorem we write

$$p_k^{n+1}(x = x^n|W^{n+1}, H^n) \quad \propto \quad f^W(W^{n+1}|x^n, \theta_k)p_k^n,$$

since we dropped the normalizing denominator when we made the transition to equation (7.61) (note that the history H^n is captured in the prior probability p_k^n on the right hand side). For this reason, we need to find the normalizing constant which we call C_w for a given observation $W^{n+1} = w$, which would be given by

$$C_w = \sum_{k=1}^{K} f^W(W^{n+1} = w|x^n, \theta_k)p_k^n.$$

Below we treat C_W as a random variable conditioned on the event that $W^{n+1} = w$. We can then write

$$p_k^{n+1}(x|W^{n+1} = w, H^n) = \frac{1}{C_w} f^W(W^{n+1} = w|x^n, \theta_k)p_k^n.$$

We note that just as $\theta^{n+1}(x)$ was a random variable when we originally stated the knowledge gradient in (7.57), the probabilities $p_k^{n+1}(x)$ are also random variables, which we see in equation (7.64) since it depends on the random outcome W^{n+1} (which depends implicitly on the experiment $x^n = x$).

This result allows us to write (7.58) as

$$\begin{aligned}
\nu^{KG}(x) = \mathbb{E}_\theta \mathbb{E}_{W|\theta} &\left\{ \max_{x'} \frac{1}{C_W} \sum_{k=1}^{K} \left(f^W(W|x^n, \theta_k)p_k^n \right) F(x', \theta_k)|S^n, x = x^n \right\}\\
&- \sum_{k=1}^{K} p_k^n F(x, \theta_k). \qquad (7.65)
\end{aligned}$$

We next have to deal with the outer expectation. Hidden in this expectation are two expectations: the expectation over the true parameter vector θ, and the random variable W^{n+1}, which depends on θ. Above, we illustrated a nonlinear model using the logistic regression equation given in (7.56), which applies to a random variable W that can take on one of two outcomes (0/1, or success/failure). For the moment, we are going to assume that W can take on one of a finite set of outcomes $w_1, \ldots, w_L$ with probability $f^W(w_\ell|x,\theta)$, where x represents our vector of inputs (that is, the decision).

The expectation can be written (keeping in mind that the entire expression is a function of x)

$$\begin{aligned}
&\mathbb{E}_\theta \mathbb{E}_{W|\theta} \left\{ \max_{x'} \frac{1}{C_W} \sum_{k=1}^{K} p_k^n f^W(W|x^n,\theta_k) F(x',\theta_k) | S^n, x = x^n \right\} \\
&= \mathbb{E}_\theta \mathbb{E}_{W|\theta} \frac{1}{C_W} \left\{ \max_{x'} \sum_{k=1}^{K} p_k^n f^W(W|x^n,\theta_k) F(x',\theta_k) | S^n, x = x^n \right\} \\
&= \sum_{j=1}^{K} \left(\sum_{\ell=1}^{L} \frac{1}{C_{w_\ell}} \left\{ \max_{x'} \sum_{k=1}^{K} p_k^n f^W(W = w_\ell|x^n,\theta_k) F(x',\theta_k) | S^n, x = x^n \right\} f^W(W = w_\ell|x,\theta_j) \right) p_j^n .
\end{aligned} \tag{7.66}$$

Equation (7.66) is computationally a little scary, especially as the number K of values of θ grows. Remember that since θ is typically a vector (and possibly a high dimensional vector), we are potentially interested in representing the set of different values of θ with a fairly large set. However, equation (7.66) has an outer sum over K, then a sum over the outcomes of W which is typically scalar (so typically not too large), then a maximization over x (which can be a relatively large set), followed by another sum over K, divided by the normalizing constant C_{w_ℓ} which also has a sum over K. If K is large (and by this we mean larger than, say, 20), then equation (7.66) starts to become expensive to compute.

The good news is that we can simplify it quite a bit.

We start by noticing that the terms $f^W(W = w_\ell|x,\theta_j)$ and p_j^n are not a function of x' or k, which means we can take them outside of the $\max$ operator. We can also reverse the order of the other sums over k and w_ℓ, giving us

$$\begin{aligned}
&\mathbb{E}_\theta \mathbb{E}_{W|\theta} \left\{ \max_{x'} \frac{1}{C_W} \sum_{k=1}^{K} p_k^n f^W(W|x^n,\theta_k) F(x',\theta_k) | S^n, x = x^n \right\} \\
&= \sum_{\ell=1}^{L} \sum_{j=1}^{K} \left(\frac{f^W(W = w_\ell|x,\theta_j) p_j^n}{C_{w_\ell}} \right) \left\{ \max_{x'} \sum_{k=1}^{K} p_k^n f^W(W = w_\ell|x^n,\theta_k) F(x',\theta_k) | S^n, x = x^n \right\} .
\end{aligned} \tag{7.67}$$

Using the definition of the normalizing constant C_w we can write

$$\begin{aligned}
\sum_{j=1}^{K} \left(\frac{f^W(W = w_\ell|x,\theta_j) p_j^n}{C_{w_\ell}} \right) &= \left(\frac{\sum_{j=1}^{K} f^W(W = w_\ell|x,\theta_j) p_j^n}{C_{w_\ell}} \right) \\
&= \left(\frac{\sum_{j=1}^{K} f^W(W = w_\ell|x,\theta_j) p_j^n}{\sum_{k=1}^{K} f^W(W = w_\ell|x,\theta_k) p_k^n} \right) \\
&= 1.
\end{aligned}$$

We just cancelled two sums over the K values of θ! This is a pretty big deal, since we will typically want K to be as large as possible (the more samples of θ we can represent, the better we can approximate the expectation over θ.

This result allows us to write (7.67) as

$$\begin{aligned}\mathbb{E}_\theta \mathbb{E}_{W|\theta} & \left\{ \max_{x'} \frac{1}{C_W} \sum_{k=1}^{K} p_k^n f^W(W|x^n, \theta_k) F(x', \theta_k) | S^n, x = x^n \right\} \\ &= \sum_{\ell=1}^{L} \left\{ \max_{x'} \sum_{k=1}^{K} p_k^n f^W(W = w_\ell | x^n, \theta_k) F(x', \theta_k) | S^n, x = x^n \right\} . \quad (7.68)\end{aligned}$$

Equation (7.68) is a very nice result, because it allows us to compute the knowledge gradient for nonlinear models using a sampled belief model in a surprisingly compact way. In practice, the sum over outcomes w_ℓ tends to be relatively compact since this is just an approximation of a scalar random variable. For example, if our random variable W is normally distributed, we would probably discretize the normal distribution into 5 or 7 ranges.

The search over x depends on how many experiments we are thinking of running, which can be large if x is a vector. However, we can again replace a search over the entire set of possible experiments $x' \in \mathcal{X}$ with a sample which just has to be large enough to help us identify which experiment x to run next.

There may be times when it is be preferable to use the sampled belief model, even when our belief model is linear. The problem with the calculations in section 7.8.2 for the linear belief model is that it depends on assuming that the distribution of belief about the parameter vector θ is multivariate normal. While this may seem elegant, in real applications this may not be very realistic. For example, we may know that a parameter θ_i only makes sense if it is positive. However, a multivariate normal distribution implies that every parameter might be negative. With a sampled belief model, we can generate samples (perhaps from a multiviate normal distribution), but reject outcomes that are felt to be inappropriate.

7.8.4 The knowledge gradient for hierarchical belief models

There are many problems where the choice "x" is multidimensional. We might be choosing a person who is described by several attributes (education, experience), the features of a product such as a laptop, or the choices made in the design of an experiment to create a new material (temperature, pressure, concentrations). In these settings, listing all possible values of x would produce an unmanageably large set of choices.

We build on the idea of using a hierarchical belief model that was originally introduced in section 3.6.1. We now want to compute the knowledge gradient to capture the expected value of information from a single experiment. Recall that the knowledge gradient is given by

$$\nu_x^{KG}(S^n) = \mathbb{E}\left[\max_{x' \in \mathcal{X}} \bar{\mu}_{x'}^{n+1}(x) | S^n\right] - \max_{x' \in \mathcal{X}} \bar{\mu}_{x'}^n. \quad (7.69)$$

Our presentation assumes a comfort with the notation for the hierarchical belief model, so we recommend returning to take a quick look at section 3.6.1.

The knowledge gradient exploits the structure of the hierarchical belief model. We first write the updating equations for the belief (mean and precision) for aggregation level g, which is given by

$$\bar{\mu}_x^{(g,n+1)} = \frac{\beta_x^{(g,n)} \bar{\mu}_x^{(g,n)} + \beta_x^W \hat{\mu}_x^{n+1}}{\beta_x^{(g,n)} + \beta_x^W}, \quad (7.70)$$

$$\beta_x^{(g,n+1)} = \beta_x^{(g,n)} + \beta_x^W, \quad (7.71)$$

Equation (7.70) can be rewritten as

$$\bar{\mu}_x^{(g,n+1)} = \bar{\mu}_x^{(g,n)} + \frac{\beta_x^W}{\beta_x^{(g,n)} + \beta_x^W} \left(\hat{\mu}_x^{n+1} - \bar{\mu}_x^{(g,n)} \right).$$

We rewrite the equation one more time as

$$\bar{\mu}_x^{(g,n+1)} = \bar{\mu}_x^{(g,n)} + \frac{\beta_x^W}{\beta_x^{(g,n)} + \beta_x^W} \left(\hat{\mu}_x^{n+1} - \bar{\mu}_x^{n} \right) + \frac{\beta_x^W}{\beta_x^{(g,n)} + \beta_x^W} \left(\bar{\mu}_x^{n} - \bar{\mu}_x^{(g,n)} \right).$$

We use a trick we have employed before by writing the updating equations in terms of the standard normal random variable Z,

$$\bar{\mu}_x^{(g,n+1)} = \bar{\mu}_x^{(g,n)} + \frac{\beta_x^W}{\beta_x^{(g,n)} + \beta_x^W} \left(\bar{\mu}_x^{n} - \bar{\mu}_x^{(g,n)} \right) + \tilde{\sigma}_x^{(g,n)} Z. \tag{7.72}$$

After some algebra, it is possible to show that the predictive variance (that is, the change in the variance of the estimate $\bar{\mu}_x^{(g,n+1)}$ given what we know after n observations) is given by

$$\tilde{\sigma}_x^{(g,n)} = \frac{\beta_x^W \sqrt{\frac{1}{\beta_x^{(g,n)}} + \frac{1}{\beta_x^W}}}{\beta_x^{(g,n)} + \beta_x^W}. \tag{7.73}$$

<table>
<tr><td>$g = 2$</td><td colspan="9">13</td></tr>
<tr><td>$g = 1$</td><td colspan="3">10</td><td colspan="3">11</td><td colspan="3">12</td></tr>
<tr><td>$g = 0$</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr>
</table>

Figure 7.10 Example with nine alternatives and three aggregation levels

We next define the sets

$\mathcal{G}(x, x')$ Set of all aggregation levels that the alternatives x and x' have in common, with $\mathcal{G}(x, x') \subseteq \mathcal{G}$. In the example in figure 7.10 we have $\mathcal{G}(2,3) = \{1, 2\}$, since $x = 2$ and $x' = 3$ have common points at aggregation levels 1 and 2.

$\mathcal{X}^{(g)}(x)$ Set of all alternatives that share the same aggregated alternative $G^{(g)}(x)$ at the g^{th} aggregation level, with $\mathcal{X}^{(g)}(x) \subseteq \mathcal{X}$. In the example in figure 7.10 we have $\mathcal{X}^1(4) = \{4, 5, 6\}$.

With the hierarchical belief model, running an experiment x can result in updated beliefs of our function at other values $x' \in \mathcal{X}$ (in fact, we might update our belief at every value, if the most aggregate level averages across the entire set). To do this, we write our original weighted estimate

$$\bar{\mu}_x^n = \sum_{g \in \mathcal{G}} w_x^{(g)} \bar{\mu}_x^{(g,n)},$$

as

$$\bar{\mu}_{x'}^{n+1} = \sum_{g \notin \mathcal{G}(x',x)} w_{x'}^{(g,n+1)} \bar{\mu}_{x'}^{(g,n+1)} + \sum_{g \in \mathcal{G}(x',x)} w_{x'}^{(g,n+1)} \bar{\mu}_x^{(g,n+1)}.$$

Using (7.72) and rearranging gives us

$$\begin{aligned}\bar{\mu}^{n+1}_{x'} &= \sum_{g\in\mathcal{G}} w^{(g,n+1)}_{x'} \bar{\mu}^{(g,n)}_{x'} + \sum_{g\in\mathcal{G}(x',x)} w^{(g,n+1)}_{x'} \frac{\beta^W_x}{\beta^{(g,n)}_x + \beta^W_x}\left(\bar{\mu}^n_x - \bar{\mu}^{(g,n)}_x\right) \\ &\quad + Z \sum_{g\in\mathcal{G}(x',x)} w^{(g,n+1)}_{x'} \tilde{\sigma}^{(g,n)}_x. \end{aligned} \tag{7.74}$$

When we run an experiment, we have to capture the fact that not only are we changing the estimates $\bar{\mu}^{(g,n)}$ for all aggregation levels g, we are also going to change the weights, which is more complicated (since these are nonlinear functions of the data). We approximate the weights using

$$\bar{w}^{(g,n)}_{x'}(x) = \frac{\left(\left(\beta^{(g,n)}_{x'} + I^{(g)}_{x',x}\beta^W_{x'}\right)^{-1} + \left(\beta^{(g,n)}_{x'}\right)^2\right)^{-1}}{\sum_{g'\in\mathcal{G}}\left(\left(\beta^{g',n}_{x'} + I^{g'}_{x',x}\beta^W_{x'}\right)^{-1} + \left(\beta^{g',n}_{x'}\right)^2\right)^{-1}}, \tag{7.75}$$

where

$$I^{(g)}_{x',x} = \left\{\begin{array}{cc} 1 & \text{if } g \in \mathcal{G}\left(x', x\right) \\ 0 & \text{otherwise} \end{array}\right. .$$

Combining (7.69) with (7.74) and (7.75), gives us the following formula for the knowledge gradient

$$\nu^{KG}_x\left(S^n\right) = \mathbb{E}\left[\max_{x'\in\mathcal{X}} a^n_{x'}(x) + b^n_{x'}(x)Z|S^n\right] - \max_{x'\in\mathcal{X}} \bar{\mu}^n_{x'}, \tag{7.76}$$

where

$$\begin{aligned} a^n_{x'}(x) &= \sum_{g\in\mathcal{G}} \bar{w}^{(g,n)}_{x'}(x)\bar{\mu}^{(g,n)}_{x'} + \sum_{g\in\mathcal{G}(x',x)} \bar{w}^{(g,n)}_{x'}(x)\frac{\beta^W_x}{\beta^{(g,n)}_x + \beta^W_x}\left(\bar{\mu}^n_x - \bar{\mu}^{(g,n)}_x\right), &(7.77)\\ b^n_{x'}(x) &= \sum_{g\in\mathcal{G}(x',x)} \bar{w}^{(g,n)}_{x'}(x)\tilde{\sigma}^{(g,n)}_x. &(7.78)\end{aligned}$$

We now build on the work in section 7.8.1 for the knowledge gradient with correlated beliefs where we represent the knowledge gradient as the maximum of a series of lines $a^n_{x'} + b^n_{x'} z$, where $a^n_{x'}$ is the mean and $b^n_{x'}$ is the predictive variance $\tilde{\sigma}^{(g,n)}_x$. We first have to identify and remove the lines that are dominated by all the rest. We refer to non-dominated lines by $\tilde{a}^n_{x'} + \tilde{b}^n_{x'} z$ over a reduced set of alternatives that are now numbered from $1, \ldots, \tilde{M}$, giving us

$$\nu^{KG,n}_x = \sum_{i=1,\ldots,\tilde{M}-1}\left(\tilde{b}^n_{i+1}(x) - \tilde{b}^n_i(x)\right) f\left(-\left|\frac{\tilde{a}^n_i(x) - \tilde{a}^n_{i+1}(x)}{\tilde{b}^n_{i+1}(x) - \tilde{b}^n_i(x)}\right|\right). \tag{7.79}$$

Recall (from our original derivation of the knowledge gradient) that

$$f(\zeta) = \zeta\Phi(\zeta) + \phi(\zeta),$$

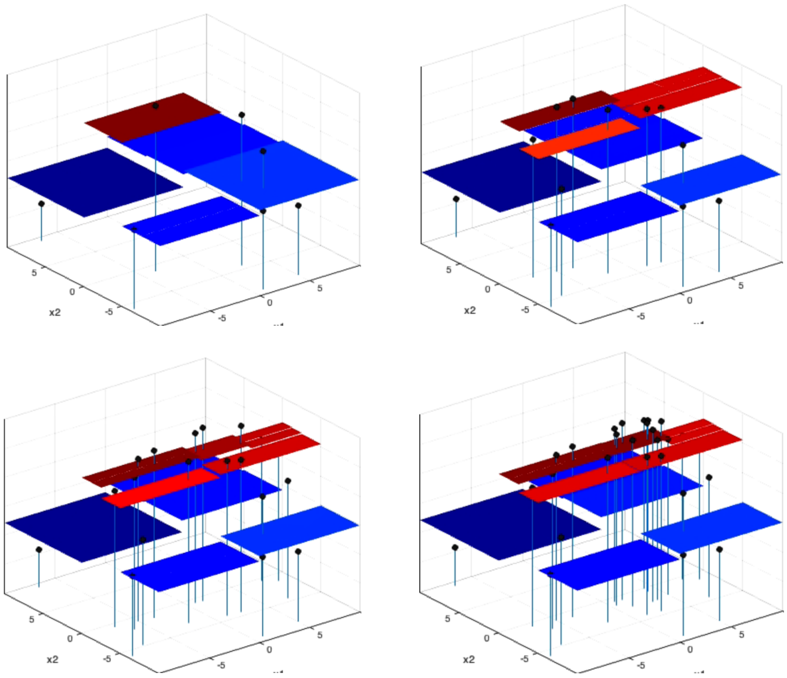


Figure 7.11 Sequence of estimates of a function using the hierarchical knowledge gradient policy, showing initial exploration to a more focused search as it identifies the optimum.

where $\Phi(\zeta)$ is the cumulative normal distribution and $\phi(\zeta)$ is the standard normal density, given respectively by

$$\Phi(\zeta) = \int_{-\infty}^{\zeta} \phi(z)dz,$$

and

$$\phi(\zeta) = \frac{1}{\sqrt{2\pi}} e^{-\frac{\zeta^2}{2}}.$$

The hierarchical knowledge gradient is an unusually powerful method for guiding experiments when the belief model is represented hierarchically. This is useful whenever an experiment x is multidimensional, which typically lends itself to this type of representation if for no other reason that we can produce aggregations by simply ignoring one dimension at a time. Figure 7.11 illustrates the behavior, progressing from initial exploration (upper left corner) to increasing focus on where it believes the optimum may be located (exploitation).

7.9 SIMULATION OPTIMIZATION

A subcommunity within the larger stochastic search community goes by the name *simulation optimization*. This community also works on problems that can be described in the form of $\max_x \mathbb{E}F(x, W)$, but the context typically arises when x represents the design of a physical system, which is then evaluated (noisily) using discrete-event simulation. The number of potential designs $\mathcal{X}$ is typically in the range of 5 to perhaps 100. The standard approach in simulation optimization is to use a frequentist belief model, where it is generally assumed that our experimental budget is large enough for us to run some initial testing of each of the alternatives to build an initial belief.

The field of simulation-optimization has its roots in the analysis of designs, such as the layout of a manufacturing system, where we can get better results if we run a discrete event simulation model for a longer time. We can evaluate a design x more accurately by increasing the run length n_x of the simulation, where n_x might be the number of time periods, the CPU time, or the number of discrete events (e.g. customer arrivals). We assume that we have a global budget N, and we need to find n_x for each x so that

$$\sum_{x \in \mathcal{X}} n_x = N.$$

For our purposes, there is no difference between a potential design of a physical system and a policy. Searching for the best design and searching for the best policy is, algorithmically speaking, identical as long as the set of policies is not too large.

We can tackle this problem using the strategies described above (such as the knowledge gradient) if we break up the problem into a series of short simulations (say, 1 time step or 1 unit of CPU time). Then, at each iteration we have to decide which design x to evaluate, contributing to our estimate θ^n_x for design x. The problem with this strategy is that it ignores the startup time for a simulation. It is much easier to set a run length n_x for each design x, and then run the entire simulation to obtain an estimate of θ_x.

The simulation-optimization problem is traditionally formulated in a frequentist framework, reflecting the lack of prior information about the alternatives. A standard strategy is to run the experiments in two stages. In the first stage, a sample n^0 is collected for each design. The information from this first stage is used to develop an estimate of the value of each design. We might learn, for example, that certain designs seem to lack any promise at all, while other designs may seem more interesting. Rather than spreading our budget across all the designs, we can use this information to focus our computing budget across the designs that offer the greatest potential.

7.9.1 An indifference zone algorithm

There are a number of algorithms that have been suggested to search for the best design using the indifference zone criterion, which is one of the most popular in the simulation-optimization community. The algorithm in figure 7.12 summarizes a method which successively reduces a set of candidates at each iteration, focusing the evaluation effort on a smaller and smaller set of alternatives. The method (under some assumptions) using a user-specified indifference zone of δ. Of course, as δ is decreased, the computational requirements increase.

7.9.2 Optimal computing budget allocation

The value of the indifference zone strategy is that it focuses on achieving a specific level of solution quality, being constrained by a specific budget. However, it is often the case that we are trying to do the best we can within a specific computing budget. For this purpose,

Step 0. Initialization:

Step 0a. Select the probability of correct selection $1-\alpha$, indifference zone parameter δ and initial sample size $n_0 \geq 2$.

Step 0b. Compute

$$\eta = \frac{1}{2}\left[\left(\frac{2\alpha}{k-1}\right)^{-2/(n_0-1)} - 1\right].$$

Step 0c. Set $h^2 = 2\eta(n_0 - 1)$.

Step 0d. Set $\mathcal{X}^0 = \mathcal{X}$ as the set of systems in contention.

Step 0e. Obtain samples W_x^m, $m = 1, \ldots, n_0$ of each $x \in \mathcal{X}^0$ and let θ_x^0 be the resulting sample means for each alternative computing using

$$\theta_x^0 = \frac{1}{n_0}\sum_{m=1}^{n_0} W_x^m.$$

Compute the sample variances for each pair using

$$\hat{\sigma}_{xx'}^2 = \frac{1}{n_0 - 1}\sum_{m=1}^{n_0}\left[W_x^m - W_{x'}^m - (\theta_x^0 - \theta_{x'}^0)\right]^2.$$

Set $r = n_0$.

Step 0f. Set $n = 1$.

Step 1. Compute

$$W_{xx'}(r) = \max\left\{0, \frac{\delta}{2r}\left(\frac{h^2\hat{\sigma}_{xx'}^2}{\delta^2} - r\right)\right\}.$$

Step 2. Refine the eligible set using

$$\mathcal{X}^n = \left\{x : x \in \mathcal{X}^{n-1} \text{ and } \theta_x^n \geq \theta_{x'}^n - W_{xx'}(r), x' \neq x\right\}.$$

Step 3. If $|\mathcal{X}^n| = 1$, stop and select the element in $\mathcal{X}^n$. Otherwise, perform an additional sample W_x^{n+1} of each $x \in \mathcal{X}^n$, set $r = r + 1$ and return to step 1.

Figure 7.12 Policy search algorithm using the indifference zone criterion, due to Nelson & Kim (2001).

a line of research has evolved under the name *optimal computing budget allocation*, or OCBA.

Figure 7.13 illustrates a typical version of an OCBA algorithm. The algorithm proceeds by taking an initial sample $N_x^0 = n_0$ of each alternative $x \in \mathcal{X}$, which means we use $B^0 = Mn_0$ experiments from our budget B. Letting $M = |\mathcal{X}|$, we divide the remaining budget of experiments $B - B^0$ into equal increments of size Δ, so that we do $N = (B - Mn_0)\Delta$ iterations.

After n iterations, assume that we have tested alternative x N_x^n times, and let W_x^m be the mth observation of x, for $m = 1, \ldots, N_x^n$. The updated estimate of the value of each alternative x is given by

$$\theta_x^n = \frac{1}{N_x^n}\sum_{m=1}^{N_x^n} W_x^m.$$

Let $x^n = \arg\max \theta_x^n$ be the current best option.

After using Mn_0 observations from our budget, at each iteration we increase our allowed budget by $B^n = B^{n-1} + \Delta$ until we reach $B^N = B$. After each increment, the allocation $N^n_x,\ x \in \mathcal{X}$ is recomputed using

$$\frac{N^{n+1}_x}{N^{n+1}_{x'}} = \frac{\hat{\sigma}^{2,n}_x / (\theta^n_{x^n} - \theta^n_{x'})^2}{\hat{\sigma}^{2,n}_{x'} / (\theta^n_{x^n} - \theta^n_{x'})^2} \quad x \neq x' \neq x^n, \tag{7.80}$$

$$N^{n+1}_{x^n} = \hat{\sigma}^n_{x^n} \sqrt{\sum_{i=1, i \neq x^n}^{M} \left(\frac{N^{n+1}_x}{\hat{\sigma}^n_i} \right)^2}. \tag{7.81}$$

We use equations (7.80)-(7.81) to produce an allocation N^n_x such that $\sum_x N^n_x = B^n$. Note that after increasing the budget, it is not guaranteed that $N^n_x \geq N^{n-1}_x$ for some x. If this is the case, we would not evaluate these alternatives at all in the next iteration. We can solve these equations by writing each N^n_x in terms of some fixed alternative (other than x^n), such as N^n_1 (assuming $x^n \neq 1$). After writing N^n_x as a function of N^n_1 for all x, we then determine N^n_1 so that $\sum N^n_x \approx B^n$ (within rounding).

The complete algorithm is summarized in figure 7.13.

7.10 LEARNING WITH LARGE OR CONTINUOUS CHOICE SETS

There are many problems where our choice set $\mathcal{X}$ is either extremely large, or continuous (which means the number of possible values is infinite). For example:

■ **EXAMPLE 7.1**

A website advertising movies has the space to show 10 suggestions out of hundreds of movies within a particular genre. The website has to choose from all possible combinations of 10 movies out of the population.

■ **EXAMPLE 7.2**

A scientist is trying to choose the best from a set of over 1000 different materials, but has a budget to only test 20.

■ **EXAMPLE 7.3**

A bakery chef for a food producer has to find the best proportions of flour, milk, yeast, and salt.

■ **EXAMPLE 7.4**

A basketball coach ha to choose the best five starting players from a team of 12. It takes approximately half a game to draw conclusions about the performance of how well five players work together.

```
Step 0. Initialization:
    Step 0a. Given a computing budget $B$, let $n^0$ be the initial sample size for each of the $M = |\mathcal{X}|$ alternatives. Divide the remaining budget $T - Mn_0$ into increments so that $N = (T - Mn_0)/\delta$ is an integer.
    Step 0b. Obtain samples $W_x^m$, $m = 1, \ldots, n_0$ samples of each $x \in \mathcal{X}$.
    Step 0c. Initialize $N_x^1 = n_0$ for all $x \in \mathcal{X}$.
    Step 0d. Initialize $n = 1$.
Step 1. Compute
        $\theta_x^n = \frac{1}{N_x^n} \sum_{m=1}^{N_x^n} W_x^m.$
    Compute the sample variances for each pair using
        $\hat{\sigma}_x^{2,n} = \frac{1}{N_x^n - 1} \sum_{m=1}^{N_x^n} (W_x^m - \theta_x^n)^2 .$
Step 2. Let $x^n = \arg\max_{x \in \mathcal{X}} \theta_x^n$.
Step 3. Increase the computing budget by $\Delta$ and calculate the new allocation $N_1^{n+1}, \ldots, N_M^{n+1}$ so that
        $\frac{N_x^{n+1}}{N_{x'}^{n+1}} = \frac{\hat{\sigma}_x^{2,n}/(\theta_{x^n}^n - \theta_{x'}^n)^2}{\hat{\sigma}_{x'}^{2,n}/(\theta_{x^n}^n - \theta_{x'}^n)^2} \quad x \neq x' \neq x^n,$
        $N_{x^n}^{n+1} = \hat{\sigma}_{x^n}^n \sqrt{\sum_{i=1, i \neq x^n}^{M} \left( \frac{N_x^{n+1}}{\hat{\sigma}_i^n} \right)^2 }.$
Step 4. Perform $\max\left(N_x^{n+1} - N_x^n, 0\right)$ additional simulations for each alternative $x$.
Step 5. Set $n = n + 1$. If $\sum_{x \in \mathcal{X}} N_x^n < B$, go to step 1.
Step 6. Return $x^n \arg\max_{x \in \mathcal{X}} \theta_x^n$.
```

Figure 7.13 Optimal computing budget allocation procedure.

Each of these examples exhibit large choice sets, particularly when evaluated relative to the budget for running experiments. Such situations are surprisingly common. We can handle these situations using a combination of strategies:

Generalized learning The first step in handling large choice sets is using a belief model that provides for a high level of generalization. This can be done using correlated beliefs for lookup table models, and parametric models, where we only have to learn a relatively small number of parameters (which we hope is smaller than our learning budget).

Sampled actions Whether we have continuous actions or large (often multidimensional) actions, we can create smaller problems by just using a sampled set of actions, just as we earlier used sampled beliefs about a parameter vector θ.

Action sampling is simply another use of Monte Carlo simulation to reduce a large set to a small one, just as we have been doing when we use Monte Carlo sampling to reduce large (often infinite) sets of outcomes of random variables to smaller, discrete sets. Thus, we might start with the optimization problem

$$F^* = \max_{x \in \mathcal{X}} \mathbb{E}_W F(x, W).$$

Often the expectation cannot be computed, so we replace the typically large set of outcomes of W, represented by some set Ω, with a sampled set of outcomes $\hat{\Omega} = \{W^1, W^2, \ldots, W^K\}$, giving us

$$\overline{F}^K = \max_{x \in \mathcal{X}} \frac{1}{K} \sum_{k=1}^{K} F(x, W^k).$$

When $\mathcal{X}$ is too large, we can play the same game and replace it with a random sample $\hat{\mathcal{X}} = \{x^1, x^2, \ldots, x^L\}$, giving us the problem

$$W^{K,L} = \max_{x \in \hat{\mathcal{X}}} \frac{1}{L} \sum_{\ell=1}^{L} F(x, W^\ell). \tag{7.82}$$

Section 4.3.3 provides results that demonstrate that the approximation $\overline{F}^K$ converges quite quickly to F^* as K increases. We might expect a similar result from $W^{K,L}$ as L increases, although there are problems where it is not possible to grow L past a certain amount (one example is given in section 7.8.3).

A strategy for overcoming this limitation is to periodically drop, say, the $L/2$ elements of $\mathcal{X}$ (based on current estimates), and then go through a process of randomly generating new values and adding them to the set until we again have L elements. We may even be able to obtain an estimate of the value of each of the new alternatives before running any new experiments. This can be done using the following:

- If we have a parametric belief model, we can estimate a value of x using our current estimate of θ. This could be a point estimate, or distribution $(p_k^n)_{k=1}^K$ over a set of possible values $\theta_1, \ldots, \theta_K$.

- If we are using lookup tables with correlated beliefs, and assuming we have access to a correlation function that gives us $Cov(F(x), F(x'))$ for any pair x and x', we can construct a belief from experiments we have run up to now. We just have to rerun the correlated belief model from chapter 3 including the new alternative, but without running any new experiments.

- We can always use nonparametric methods (such as kernel regression) to estimate the value of any x from the observations we have made so far, simply by smoothing over the new point. Nonparametric methods can be quite powerful (hierarchical aggregation is an example, even though we present it alongside lookup table models in chapter 3), but they assume no structure and as a result need more observations.

Using these estimates, we might require that any newly generated alternative x be at least as good as any of the estimates of values in the current set. This process might stop if we cannot add any new alternatives after testing some number M.

7.11 RESTLESS BANDITS AND NONSTATIONARY LEARNING

Our basic model assumes that we are sequentially observing different alternatives x where we get to make noisy observations of an unknown truth μ_x, which we write as

$$W_t = \mu_x + \varepsilon_t.$$

When we use a lookup table belief model, we can use W_{t+1} to update our belief $\bar{\mu}_{tx}$ to obtain $\bar{\mu}_{t+1,x}$. Observing x changes our belief about $\bar{\mu}_x$, and possibly about $\bar{\mu}_{x'}$ if we have correlated beliefs. However, the truth is unchanging.

There are many problems where the truth is, in fact, changing. This problem is known in the learning literature as "restless bandits" since each "bandit" (more precisely, each arm) returns a reward from a distribution that is evolving over time. This problem has attracted considerable interest from the research community looking to establish theoretical properties of different policies. Our presentation focuses on creating the model, which we can then solve using any of our four classes of policies.

Below, we first present a model of a learning problem with transient truths. We then describe how to compute the knowledge gradient, and close with a discussion of how to use other policies for this setting.

7.11.1 A transient learning model

We first introduced this model in section 3.11 where the true mean varies over time according to the model

$$\mu_{t+1} = M_t \mu_t + \delta_{t+1},$$

where δ_{t+1} is a random variable with distribution $N(0, \sigma_\delta^2)$, which means that $\mathbb{E}\{\mu_{t+1}|\mu_t\} = \mu_t$. Recall that M_t is a diagonal matrix that captures predictable changes (e.g. where the means are increasing or decreasing predictably). If we let M_t be the identity matrix, then we have the simpler problem where the changes in the means have mean 0 which means that we expect $\mu_{t+1} = \mu_t$. However, there are problems where there can be a predictable drift, such as estimating the level of a reservoir changing due to stochastic rainfall and predictable evaporation. We then make noisy observations of μ_t using

$$W_t = M_t \mu_t + \varepsilon_t.$$

In our optimization setting, we get to choose the element x we want to observe. It used to be that if we did not observe an alternative x' that our belief $\bar{\mu}_{tx'}$ did not change (and of course, nor did the truth). Now, the truth may be changing, and to the extent that there is predictable variation (that is, M_t is not the identity matrix), then even our beliefs may change.

The updating equation for the mean vector is given by

$$\bar{\mu}_{t+1,x} = \begin{cases} M_{tx}\bar{\mu}_{tx} + \frac{W_{t+1} - M_{tx}\bar{\mu}_{tx}}{\sigma_\varepsilon^2 + \Sigma_{t,xx}} \Sigma_{t,xx} & \text{If } x_t = x, \\ M_{tx}\bar{\mu}_{tx} & \text{Otherwise.} \end{cases} \tag{7.83}$$

To describe the updating of Σ_t, let Σ_{tx} be the column associated with alternative x, and let e_x be a vector of 0's with a 1 in the position corresponding to alternative x. The updating equation for Σ_t can then be written

$$\Sigma_{t+1,x} = \begin{cases} \Sigma_{tx} - \frac{\Sigma_{tx}^T \Sigma_{tx}}{\sigma_\varepsilon^2 + \Sigma_{t,xx}} e_x & \text{If } x_t = x, \\ \Sigma_{tx} & \text{Otherwise.} \end{cases} \tag{7.84}$$

These updating equations can play two roles in the design of learning policies. First, they can be used in a lookahead policy, as we illustrate next with the knowledge gradient (a one-step lookahead policy). Alternatively, they can be used in a simulator for the purpose of doing policy search for the best PFA or CFA.

7.11.2 The knowledge gradient for transient problems

To compute the knowledge gradient, we first compute

$$\begin{aligned}
\tilde{\sigma}^2_{tx} &= \text{The conditional change in the variance of } \bar{\mu}_{t+1,x} \text{ given what we know now,} \\
&= \mathit{Var}(\bar{\mu}_{t+1,x}|\bar{\mu}_t) - \mathit{Var}(\bar{\mu}_t), \\
&= \tilde{\Sigma}_{t,xx}.
\end{aligned}$$

We can use $\tilde{\sigma}_{tx}$ to write the updating equation for $\bar{\mu}_t$ using

$$\bar{\mu}_{t+1} = M_t\bar{\mu}_t + \tilde{\sigma}_{tx}Z_{t+1}e_p,$$

where $Z_{t+1} \sim N(0,1)$ is a scalar, standard normal random variable.

We now present some calculations that parallel the original knowledge gradient calculations. First, we define ζ_{tx} as we did before

$$\zeta_{tx} = -\left|\frac{\bar{\mu}_{tx} - \max_{x' \neq x}\bar{\mu}_{tx'}}{\tilde{\sigma}_{tx}}\right|.$$

This is defined for our stationary problem. We now define a modified version that we call ζ^M_{tx} that is given by

$$\zeta^M_{tx} = M_t\zeta_{tx}.$$

We now compute the knowledge gradient for nonstationary truths using a form that closely parallels the original knowledge gradient,

$$\begin{aligned}
\nu^{KG-NS}_{tx} &= \tilde{\sigma}_{tx}\left(\zeta^M_{tx}\Phi(\zeta^M_{tx}) + \phi(\zeta^M_{tx})\right) &\quad (7.85)\\
&= \tilde{\sigma}_{tx}\left(M_t\zeta_{tx}\Phi(M_t\zeta_{tx}) + \phi(M_t\zeta_{tx})\right). &\quad (7.86)
\end{aligned}$$

It is useful to compare this version of the knowledge gradient to the knowledge gradient for our original problem with static truths. If M_t is the identity matrix, then this means that the truths μ_t are not changing in a predictable way; they might increase or decrease, but on average μ_{t+1} is the same as μ_t. When this happens, the knowledge gradient for the transient problem is the same as the knowledge gradient when the truths are not changing at all.

So, does this mean that the problem where the truths are changing is the same as the one where they remain constant? Not at all. The difference arises in the updating equations, where the precision of alternatives x' that are not tested decrease, which will make them more attractive from the perspective of information collection.

7.11.3 Policy search for transient problems

In section 7.3 we introduced the idea of searching within a class of policies we call policy function approximations (PFAs) or cost function approximations (CFAs), which are parameterized policies that can be optimized to optimize either the final reward (equation (7.2)) or cumulative reward (equation (7.3)).

Policy search requires only that we be able to simulate a policy (we describe the process of tuning policies below). The process of tuning policies requires only that we be able to simulate the underlying dynamics, regardless of whether the truth is stationary or transient.

Problem	IE	UCBE
Goldstein	0.0099	2571
AUF_HNoise	0.0150	0.319
AUF_MNoise	0.0187	1.591
AUF_LNoise	0.01095	6.835
Branin	0.2694	.000366
Ackley	1.1970	1.329
HyperEllipsoid	0.8991	21.21
Pinter	0.9989	0.000164
Rastrigin	0.2086	0.001476

Table 7.6 Optimal tuned parameters for interval estimation IE, and UCB-E (from Wang et al. (2016)).

The basic structure of policies does not change just because we are working in a transient environment, but a transient truth will affect the tuning of policies, and may even affect the decision of which class of policy works best. In particular, we would expect a transient environment to encourage greater exploration.

7.12 TUNING POLICIES

An important issue when designing policies is the process of tuning. Upper confidence bounding policies, for example, enjoy elegant finite-time bounds, but in practice, these policies only work well when they are tuned.

7.12.1 The effect of scaling

Consider the case of two policies. The first is interval estimation, given by

$$X^{IE}(S^n|\theta^{IE}) = \arg\max_x \left(\bar{\mu}_x^n + \theta^{IE}\sigma_x^n\right),$$

which exhibits a unitless tunable parameter θ^{IE}. The second policy is a type of upper confidence bounding policy known in the literature as UCB-E, given by

$$X^{UCB-E,n}(S^n|\theta^{UCB-E}) = \arg\max_x \left(\bar{\mu}_x^n + \sqrt{\frac{\theta^{UCB-E}}{N_x^n}}\right),$$

where N_x^n is the number of times that we have evaluated alternative x. We note that unlike the interval estimation policy, the tunable parameter θ^{UCB-E} has units, which means that we have to search over a much wider range than we would when optimizing θ^{IE}.

Each of these parameters were tuned on a series of benchmark learning problems using the testing system MOLTE (described below). We see that the optimal value of θ^{IE} ranges from around 0.01 to 1.2. By contrast, θ^{UCB-E} ranges from 0.0001 to 2500.

These results illustrate the effect of units on tunable parameters. The UCB-E policy enjoys finite time bounds on its regret, but would never produce reasonable results without tuning. By contrast, the optimal values of θ^{IE} for interval estimation vary over a narrower range, although conventional wisdom for this parameter is that it should range between around 1 and 3. If θ^{IE} is small, then the IE policy is basically a pure exploitation policy.

Parameter tuning can be difficult in practice. Imagine, for example, an actual setting where an experiment is expensive. How would tuning be done? This issue is typically ignored in the research literature where standard practice is to focus on provable qualities. We argue that despite the presence of provable properties, the need for parameter tuning is the hallmark of a heuristic. If tuning cannot be done, the actual empirical performance of a policy may be quite poor.

Bayesian policies such as the knowledge gradient do not have tunable parameters, but do require the use of priors. Just as we do not have any real theory to characterize the behavior of algorithms that have (or have not) been tuned, we do not have any theory to describe the effect of incorrect priors.

7.12.2 MOLTE - Optimal learning testing system

MOLTE (Modular, Optimal Learning Testing Environment) is a Matlab-based system that is designed to help researchers evaluate learning policies on a wide range of problems. Each policy is encoded in its own `.m` file. Each test problem is also encoded in its own `.m` file. The user can specify which policies to test on which problems through a simple spreadsheet. It is quite easy to add new test problems, and new policies.

MOLTE then provides a wide range of reports to allow people to understand how the different policies behave. This software is available on the internet at `http://castlelab.princeton.edu/software.htm#molte`.

MOLTE has been used to compare a variety of learning policies on a series of benchmark problems which are characterized by low-dimensional, discretized decision vectors x. A series of policies were compared to the knowledge gradient policy for online learning, using a correlated belief model (reflecting the property that we are learning continuous surfaces). Online KG was compared to interval estimation (IE), upper confidence bounding using the UCB-E policy, the UCB-V policy, Kriging, Thompson sampling (TS), and pure exploration.

The results are shown in table 7.7, which reports the opportunity cost (OC) relative to online KG (negative means the policy outperformed online KG), and the probability that the policy outperforms online KG over 1000 simulations. Any policy with a tunable parameter is optimized by MOLTE.

The results suggest that online KG generally works quite well, but it is important to recognize that it is also computationally much more expensive than the other policies, which limits its usefulness in high-speed applications such as the internet (e.g. finding the next product to display in an electronic marketplace), where computational budgets are quite limited. We would also note that while policies such as UCB-E and UCB-V enjoy nice regret bounds, they do not perform as well. By contrast, interval estimation actually works the best, and yet IE is a policy that does not seem to enjoy any theoretical properties.

7.13 LEARNING PROBLEMS WITH ADDITIONAL STATE VARIABLES

This chapter has focused on stochastic optimization problems where the state S^n consists purely of belief state variables, which means they capture our belief about a function such as $\mathbb{E}F(x, W)$. This is the class we refer to as "state independent problems" which means

Problem Class	IE		UCBE		UCBV		Kriging		TS		EXPL	
	OC	Prob.	OC	Prob.	OC	Prob.	OC	Prob.	OC	Prob.	OC	Prob.
Goldstein	-0.061	0.81	-0.097	0.92	-0.003	0.45	-0.031	0.73	0.100	0.09	0.041	0.16
AUF_HNoise	0.058	0.40	0.022	0.43	0.037	0.54	0.031	0.39	0.073	0.22	0.047	0.48
AUF_MNoise	0.043	0.29	0.027	0.42	0.343	0.21	0.023	0.28	0.173	0.21	-0.057	0.52
AUF_LNoise	-0.043	0.73	-0.013	0.64	0.053	0.51	0.005	0.53	0.038	0.20	0.003	0.62
Branin	-0.027	0.76	0.025	0.24	0.026	0.26	0.004	0.54	0.041	0.07	0.123	0.00
Ackley	0.007	0.42	0.04	0.41	0.106	0.20	0.037	0.42	0.100	0.23	0.344	0.00
HyperEllipsoid	-0.059	0.73	0.064	0.12	0.08	0.07	0.146	0.22	0.011	0.38	0.243	0.03
Pinter	-0.028	0.56	-0.003	0.51	0.029	0.42	-0.055	0.65	0.122	0.19	0.177	0.04
Rastrigin	-0.082	0.70	-0.03	0.56	0.162	0.04	-0.026	0.57	0.136	0.08	0.203	0.01

Table 7.7 Comparisons with the online knowledge gradient for correlated beliefs. Negative OC means that the policy outperformed online KG (from Wang et al. (2016)).

that the function $\mathbb{E}F(x,W)$ does not depend on the state S^n. Rather, S^n only contains information about $\mathbb{E}F(x,W)$.

Starting in chapter 8, we are going to make the transition to a much richer array of problems, but we are going to give a hint of this broader class by discussing additional state information in the context of the derivative-free stochastic optimization problems that are the focus of this chapter. We are going to begin with a discussion of an important problem class in the learning literature where information about the function we are optimizing is revealed before we make our decision. This problem class has received considerable attention in the multiarmed bandit literature under the name of *contextual bandits*. We then build

7.13.1 Stochastic optimization with exogenous state information

The original statement of our basic stochastic optimization problem (in its asymptotic form),

$$\max_x \mathbb{E}F(x,W).$$

is looking for a solution in the form of a deterministic decision x^*. We then proposed that a better form was

$$\max_x \mathbb{E}\{F(x,W)|S_0\}. \tag{7.87}$$

Again, we assume that we are looking for a single decision x^*, although now we have to recognize that technically, this decision is a function of the initial state S_0.

Now consider an adaptive learning process where a new initial state S_0 is revealed each time we try to evaluate $F(x,W)$. This changes the learning process, since each time we observe $F(x,W)$ for some x and a sampled W, what we learn has to reflect that it is in the context of the initial state S_0. Some illustrations of this setting are:

■ **EXAMPLE 7.1**

Consider a newsvendor problem where S_0 is the weather forecast for tomorrow. We know that if it is raining or very cold, that sales will be lower. We need to find an optimal order decision that reflects the weather forecast. Given the forecast, we make a decision of how many newspapers to stock, and then observe the sales.

■ **EXAMPLE 7.2**

A patient arrives to a hospital with a complaint, and a doctor has to make treatment decisions. The attributes of the patient represent initial information that the patient provides in the form of a medical history, then a decision is made, followed by a random outcome (the success of the treatment).

In both of these examples, we have to make our decision given advance information (the weather, or the attributes of the patient). Instead of finding a single optimal solution x^*, we need to find a function $x^*(S_0)$. This function is a form of policy (since it is a mapping of state to action).

This problem was first studied as a type of multiarmed bandit problems, which we first introduced in chapter 2. In this community, these are known as *contextual bandit problems*, but as we show here, when properly modeled this problem is simply an instance of a sequential decision problem.

We propose the following model of contextual problems. First, we let B_t be our belief state at time t that captures our belief about the function $F(x) = \mathbb{E}F(x, W)$ (keep in mind that this is distributional information). We then model two types of exogenous information. The first we call W_t^e which is exogenous information that arrives before we make a decision (this would be the weather in our newsvendor problem, or the attributes of the patient before making the medical decision). Then, we let W_{t+1}^o be the exogenous information that captures the outcome of the decision after the decision x_t. The exogenous outcome W_t^o, along with the decision x_t and the information (B_t and W_t^e), is used to produce an updated belief state B_{t+1}.

Using this notation, the sequencing of information, belief states and decisions is

$$(B_0, W_0^e, x_0, W_1^o, B_1, W_1^e, x_1, W_2^o, B_2, \ldots).$$

We have written the sequence (W_t^o, B_t, W_t^e) to reflect the logical progression where we first learn the outcome of a decision W_t^o, then update our belief state producing B_t, and then observe the new exogenous information W_t^e before making decision x_t. However, we can write $W_t = (W_t^o, W_t^e)$ as the exogenous information, which leads to a new state $S_t = (B_t, W_t^e)$. Our policy $X_t^\pi(S_t)$ will depend on both our belief state B_t about $\mathbb{E}F(x, W)$, as well as the new exogenous information. This change of variables, along with defining $S_0 = (B_0, W_0^e)$, gives us our usual sequence of states, actions and new information. We then have to search over policies (as in (4.4)) for problems with cumulative rewards.

There is an important difference between this problem and the original terminal reward problem. In that problem, we had to find the best policy to collect information to help us find a deterministic implementation decision after our budget of N experiments is depleted, given by $x^{\pi,N}$. When we introduce the exogenous state information, it means we have to find a policy to collect information, but now we are using this information to learn a function $x^{\pi,N}(W^e)$ which we recognize is a form of policy. If we are addressing the problem of maximizing cumulative rewards over time, we need to find a policy $X^\pi(S_t) = X^\pi(B_t, W_t^e)$ that balances exploration and exploitation as we encountered with all other cumulative reward problems.

Thus, we see again that a seemingly new problem class is simply another instance of a sequential learning problem.

7.13.2 State-dependent vs. state-independent problems

Imagine that we are running experiments on a set of discrete alternatives $x \in \mathcal{X}$. We have previously introduced the idea of a belief state B^n which captures what we know (or believe) about the performance of each choice x. If we are using a frequentist perspective, our belief state can be represented using

$$B^n = \left(\bar{\mu}^n_x, \sigma^{2,n}_x, N^n_x\right)_{x\in\mathcal{X}}.$$

If we are using a Bayesian belief model, then we would represent our belief state using

$$B^n = \left(\bar{\mu}^n_x, \sigma^{2,n}_x\right)_{x\in\mathcal{X}}.$$

If we are using a parametric model with a linear architecture (with a frequentist perspective), our belief state would be given by

$$B^n = (\bar{\mu}^n, B^n).$$

If we are using a nonparametric model, the belief state is literally the entire history of observations,

$$B^n = (W^i, x^i)^n_{i=1}.$$

We have seen an array of problems in this chapter where the belief state B^n is the same as "the state" S^n. In all of these problems, the state is the belief state that is only used in the policy and does not affect the problem we are trying to solve.

Now consider our *problem* which we write as

$$\max_{x\in\mathcal{X}_t} \mathbb{E}_{S_t} E_{W|S_t}\{C(S_t, x, W)|S_t\}$$

where S_t might contain information describing a probability distribution about an uncertain parameter, such as the response of demand to price, or the relationship between the dosage of a diabetes medication and its ability to lower blood sugar. The constraints $\mathcal{X}_t$ might reflect the inventory (of blood, energy or money) on hand at time t, and the contribution function $C(S_t, x, W)$ may depend on dynamic prices, temperatures, or unemployment rates. None of these characteristics applied to the problems we considered in this chapter, but clearly instances of these "state-dependent problems" arises in a wide range of applications.

One of the most common classes of "state-dependent problems" involves the management of physical resources. This arises in the management of inventories, scheduling people and equipment, managing robots and drones, and the physical collection of information using sensors or technical experts (e.g. doctors examining patients for the spread of disease). It also applies to the management of financial resources, as might arise in the management of financial portfolios, or simply whether to hold or sell an asset. Physical states are typically reflected in the constraints, since there are many applications where decisions only impact physical resources (although this is not uniformly the case).

Interestingly, the presence of dynamic data of any type, regardless of whether it impacts the expectation, the constraints or the objective function itself, moves us into an entirely new problem class that we refer to as *state-dependent problems* which we address starting in chapter 8, and which is the focus of the entire rest of the book. In fact, it is these problem classes that are typically envisioned when someone refers to a "dynamic program," although as we now see, even the state-independent problems we addressed in this chapter (or chapter 5) are also "dynamic programs." However, when we design algorithms (or policies) for these problems, state-dependent problems introduce fresh computational challenges.

7.14 BIBLIOGRAPHIC NOTES

Section 7.3 - A nice introduction to various learning strategies is contained in Kaelbling (1993) and Sutton & Barto (1998). Thrun (1992) contains a good discussion of exploration in the learning process. The discussion of Boltzmann exploration and epsilon-greedy exploration is based on Singh et al. (2000). Interval estimation is due to Kaelbling (1993). The upper confidence bound is due to Lai & Robbins (1985). We use the version of the UCB rule given in Lai (1987). The UCB1 policy is given in **?**. Analysis of UCB policies are given in Lai & Robbins (1985) and **?**, as well as **?**.

Section 7.6 - What came to be known as "Gittins indices" was first introduced in Gittins & Jones (1974) to solve bandit problems (see DeGroot (1970) for a discussion of bandit problems before the development of Gittins indices). This was more thoroughly developed in Gittins (1979), Gittins (1981), and Gittins (1989). Whittle (1982) and Ross (1983) provide very clear tutorials on Gittins indices, helping to launch an extensive literature on the topic (see, for example, Lai & Robbins (1985), Berry & Fristedt (1985), and **?**). The work on approximating Gittins indices is due to **?**, Yao (2006) and **?**.

Section 7.7.2 - The knowledge gradient policy for normally distributed rewards and independent beliefs was introduced by Gupta & Miescke (1996), and subsequently analyzed in greater depth by Powell et al. (2008). The knowledge gradient for correlated beliefs was introduced by Frazier et al. (2009). The adaptation of the knowledge gradient for online problems is due to Ryzhov & Powell (2009).

PROBLEMS

7.1 Consider the problem of finding the best in a set of discrete choices $\mathcal{X} = \{x_1, \ldots, x_M\}$. Assume that for each alternative you maintain a lookup table belief model, where $\bar{\mu}^n_x$ is your estimate of the true mean μ_x, with precision β^n_x. Assume that your belief about μ_x is Gaussian, and let $X^\pi(S^n)$ be a policy that specifies the experiment $x^n = X^\pi(S^n)$ that you will run next, where you will learn $W^{n+1}_{x^n}$ which you will use to update your beliefs.

a) (10 points) Formulate this learning problem as a stochastic optimization problem. Define your state variable, decision variable, exogenous information, transition function and objective function.

b) (5 points) Specify three possible policies, with no two from the same policy class (PFA, CFA, VFA and DLA).

7.2 Assume that we have a standard normal prior about a true parameter μ which we assume is normally distributed with mean $\bar{\mu}^0$ and variance $(\sigma^0)^2$.

a) Given the observations $W^1, \ldots, W^n$, is $\bar{\mu}^n$ deterministic or random?

b) Given the observations $W^1, \ldots, W^n$, what is $\mathbb{E}(\mu|W^1, \ldots, W^n)$ (where μ is our truth)? Why is μ random given the first n experiments?

c) Given the observations $W^1, \ldots, W^n$, what is the mean and variance of $\bar{\mu}^{n+1}$? Why is $\bar{\mu}^{n+1}$ random?

7.3 Table 7.9 shows the priors $\bar{\mu}^n$ and the standard deviations σ^n for five alternatives.

a) Three of the alternatives have the same standard deviation, but with increasing priors. Three have the same prior, but with increasing standard deviations. Using only this information, state any relationships that you can between the knowledge gradients for each alternative. Note that you will not be able to completely rank all the alternatives.

b) Compute the knowledge gradient for each alternative assuming that $\sigma^W = 4$.

Choice	$\bar{\mu}^n$	σ^n
1	3.0	8.0
2	4.0	8.0
3	5.0	8.0
4	5.0	9.0
5	5.0	10.0

Table 7.8 Priors for exercise 9.0

7.4 There are seven alternatives with normally distributed priors on μ_x for $x \in \{1, 2, 3, 4, 5, 6, 7\}$ given in the table below:

Choice	$\bar{\mu}^n$	σ^n
1	5.0	9.0
2	3.0	8.0
3	5.0	10.0
4	4.5	12.0
5	5.0	8.0
6	5.5	6.0
7	4.0	8.0

Table 7.9 Priors

Without doing any calculations, state any relationships between the alternatives based on the knowledge gradient. For example, $1 < 2 < 3$ means 3 has a higher knowledge gradient than 2 which is better than 1 (if this was the case, you do not have to separately say that $1 < 3$).

7.5 You have to find the best of five alternatives. After n experiments, you have the data given in the table below. Assume that the precision of the experiment is $\beta^W = 0.6$.

Choice	θ^n	β^n	β^{n+1}	$\tilde{\sigma}$	$\max_{x' \neq x} \theta^n_{x'}$	ζ	$f(\zeta)$	ν^{KG}_x
1	3.0	0.444	1.044	1.248	6	-2.404	0.003	0.003
2	5.0	0.160	0.760	2.321	6	-0.431	0.220	0.511
3	6.0	0.207	0.807	2.003	5	-0.499	0.198	0.397
4	4.0	0.077	?	?	?	?	?	?
5	2.0	0.052	0.652	4.291	6	-0.932	0.095	0.406

a) Give the definition of the knowledge gradient, first in plain English and second using mathematics.

b) Fill in the missing entries for alternative 4 in table 7.5. Be sure to clearly write out each expression and then perform the calculation. For the knowledge gradient ν^{KG}_x, you will need to use a spreadsheet (or Matlab) to compute the normal distribution.

c) Now assume that we have an online learning problem. We have a budget of 20 experiments, and the data in the table above shows what we have learned after three experiments. Assuming no discounting, what is the online knowledge gradient for alternative 2? Give both the formula and the number.

7.6 You have to find the best of five alternatives. After n experiments, you have the data given in the table 7.6 below. Assume that the precision of the experiment is $\beta^W = 0.6$.

Alternative	$\bar{\mu}^n$	$\bar{\sigma}^n$	$\tilde{\sigma}$	ζ	$f(\zeta)$	KG index
1	4.0	2.5	2.321	-0.215	0.300	0.696
2	4.5	3.0	?	?	?	?
3	4.0	3.5	3.365	-0.149	0.329	1.107
4	4.2	4.0	3.881	-0.077	0.361	1.401
5	3.7	3.0	2.846	-0.281	0.274	0.780

a) Give the definition of the knowledge gradient, first in plain English and second using mathematics.

b) Fill in the missing entries for alternative 2 in table 7.6. Be sure to clearly write out each expression and then perform the calculation. For the knowledge gradient ν^{KG}_x, you will need to use a spreadsheet (or Matlab) to compute the normal distribution.

c) Now assume that we have an online learning problem. We have a budget of 20 experiments, and the data in the table above shows what we have learned after three experiments. Assuming no discounting, what is the online knowledge gradient for alternative 2? Give both the formula and the number.

7.7 You have three alternatives, with priors (mean and precision) as given in the first line of the table below. You then observe each of the alternatives in three successive experiments, with outcomes shown in the table. All observations are made with precision $\beta^W = 0.2$. Assume that beliefs are independent.

Iteration	A	B	C
Prior (μ_x^0, β_x^0)	(32,0.2)	(24,0.2)	(27,0.2)
1	36	-	-
2	-	-	23
3	-	22	-

Table 7.10 Three observations, for three alternatives, given a normally distributed belief, and assuming normally distributed observations.

a) (5 points) Give the objective function (algebraically) for offline learning (maximizing terminal reward) if you have a budget of three experiments, and where you evaluate the policy using the truth (as you would do in a simulator).

b) (5 points) Give the numerical value of the policy that was used to generate the choices that created table 7.10, using our ability to use the simulated truth (as you have done in your homeworks). This requires minimal calculations (which can be done without a calculator).

c) (10 points) Now assume that you need to run experiments in an online (cumulative reward) setting. Give the objective function (algebraically) to find the optimal policy for online learning (maximizing cumulative reward) if you have three experiments. Using the numbers in the table, give the performance of the policy that generated the choices that were made. (This again requires minimal calculations.)

7.8 Section 7.3 introduces four classes of policies for derivative-free stochastic search, a concept that was not discussed when we introduced derivative-based stochastic search in chapter 5. In which of the four classes of policies would you classify a stochastic gradient algorithm? Explain, and describe a key step in the design of stochastic gradient algorithms that is explained by your choice of policy class.

7.9 What is the relationship between the deterministic regret $R^{static,\pi}$ (recall that this was done for a machine learning problem where the "decision" is to choose a parameter θ) in equation (7.22) and the regret $R^{\pi,n}(\omega)$ for a single sample path ω in equation (7.24)? Write the regret $R^{\pi,n}(\omega)$ in equation (7.24) in the context of a learning problem and explain what is meant by a sample ω.

7.10 What is the relationship between the expected regret $\mathbb{E}R^{\pi,n}$ in equation (7.25) and the pseudo-regret $\bar{R}^{\pi,n}$ in equation (7.25)? Is one always at least as large as the other? Describe a setting under which each would be appropriate.

7.11 Figure 7.14 shows the belief about an unknown function as three possible curves, where one of the three curves is the true function. Our goal is to find the point x^* that maximizes the function. Without doing any computation (or math), create a graph and draw the general shape of the knowledge gradient for each possible experiment x. [Hint: the knowledge gradient captures your ability to make a better decision using more information.]

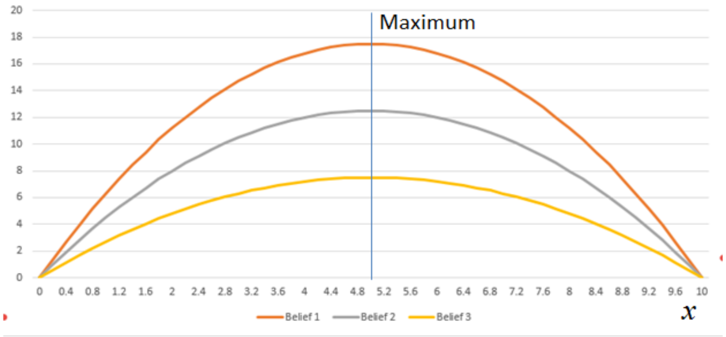


Figure 7.14 Use to plot the shape of the knowledge gradient for all x.

7.12 Assume you are trying to find the best of five alternatives. The actual value μ_x, the initial estimate $\bar{\mu}^0_x$ and the initial standard deviation $\bar{\sigma}^0_x$ of each $\bar{\mu}^0_d$ are given in table 7.11. [This exercise does not require any numerical work.]

a) Consider the following learning policies:

(1) Pure exploitation.

(2) Interval exploration.

(3) The upper confidence bounding (pick any variant).

(4) Thompson sampling.

(5) The knowledge gradient.

Write out each policy and identify any tunable parameters. How would you go about tuning the parameters?

b) Classify each of the policies above as a i) Policy function approximation (PFA), ii) Cost function approximation (CFA), iii) Policy based on a value function approximation (VFA), or iv) Direct lookahead approximation (DLA).

c) Set up the optimization formulation that can serve as a basis for evaluating these policies in an online (cumulative reward) setting (just one general formulation is needed - not one for each policy).

Alternative	μ	$\bar{\mu}^0$	$\bar{\sigma}^0$
1	1.4	1.0	2.5
2	1.2	1.2	2.5
3	1.0	1.4	2.5
4	1.5	1.0	1.5
5	1.5	1.0	1.0

Table 7.11 Prior beliefs for learning exercise

7.13 Joe Torre, former manager of the great Yankees, had to struggle with the constant game of guessing who his best hitters are. The problem is that he can only observe a hitter if he puts him in the order. He has four batters that he is looking at. The table below shows their actual batting averages (that is to say, batter 1 will produce hits 30 percent of the time, batter 2 will get hits 32 percent of the time, and so on). Unfortunately, Joe does not know these numbers. As far as he is concerned, these are all .300 hitters.

For each at-bat, Joe has to pick one of these hitters to hit. Table 7.12 below shows what would have happened if each batter were given a chance to hit (1 = hit, 0 = out). Again, Joe does not get to see all these numbers. He only gets to observe the outcome of the hitter who gets to hit.

Assume that Joe always lets the batter hit with the best batting average. Assume that he uses an initial batting average of .300 for each hitter (in case of a tie, use batter 1 over batter 2 over batter 3 over batter 4). Whenever a batter gets to hit, calculate a new batting average by putting an 80 percent weight on your previous estimate of his average plus a 20 percent weight on how he did for his at-bat. So, according to this logic, you would choose batter 1 first. Since he does not get a hit, his updated average would be $0.80(.200) + .20(0) = .240$. For the next at-bat, you would choose batter 2 because your estimate of his average is still .300, while your estimate for batter 1 is now .240.

After 10 at-bats, who would you conclude is your best batter? Comment on the limitations of this way of choosing the best batter. Do you have a better idea? (It would be nice if it were practical.)

	Actual batting average			
	0.300	0.320	0.280	0.260
Day	Batter			
	A	B	C	D
1	0	1	1	1
2	1	0	0	0
3	0	0	0	0
4	1	1	1	1
5	1	1	0	0
6	0	0	0	0
7	0	0	1	0
8	1	0	0	0
9	0	1	0	0
10	0	1	0	1

Table 7.12 Data for problem 7.13

7.14 There are four paths you can take to get to your new job. On the map, they all seem reasonable, and as far as you can tell, they all take 20 minutes, but the actual times vary quite a bit. The value of taking a path is your current estimate of the travel time on that path. In the table below, we show the travel time on each path if you had travelled that path. Start with an initial estimate of each value function of 20 minutes with your tie-breaking rule to use the lowest numbered path. At each iteration, take the path with the best estimated value, and update your estimate of the value of the path based on your experience. After 10 iterations, compare your estimates of each path to the estimate you obtain by averaging the "observations" for each path over all 10 days. Use a constant stepsize of 0.20. How well did you do?

	Paths			
Day	1	2	3	4
1	37	29	17	23
2	32	32	23	17
3	35	26	28	17
4	30	35	19	32
5	28	25	21	26
6	24	19	25	31
7	26	37	33	30
8	28	22	28	27
9	24	28	31	30
10	33	29	17	29

7.15 Assume you are considering five options. The actual value μ_d, the initial estimate $\bar{\mu}_d^0$ and the initial standard deviation $\bar{\sigma}_d^0$ of each $\bar{\mu}_d^0$ are given in table 7.13. Perform 20 iterations of each of the following algorithms:

(a) Gittins exploration using $\Gamma(n) = 2$.

(b) Interval estimation using $\theta^{IE} = 2$.

(c) The upper confidence bound algorithm using $\theta^{UCB} = 6$.

(d) The knowledge gradient algorithm.

(e) Pure exploitation.

(f) Pure exploration.

Each time you sample a decision, randomly generate an observation $W_d = \mu_d + \sigma^\varepsilon Z$ where $\sigma^\varepsilon = 1$ and Z is normally distributed with mean 0 and variance 1. [Hint: You can generate random observations of Z in Excel by using *=NORMSINV(RAND())*.]

7.16 Repeat exercise 7.15 using the data in table 7.14, with $\sigma^\varepsilon = 10$.

7.17 Repeat exercise 7.15 using the data in table 7.15, with $\sigma^\varepsilon = 20$.

Decision	μ	$\bar{\theta}^0$	$\bar{\sigma}^0$
1	1.4	1.0	2.5
2	1.2	1.2	2.5
3	1.0	1.4	2.5
4	1.5	1.0	1.5
5	1.5	1.0	1.0

Table 7.13 Data for exercise 7.15

Decision	μ	$\bar{\theta}^0$	$\bar{\sigma}^0$
1	100	100	20
2	80	100	20
3	120	100	20
4	110	100	10
5	60	100	30

Table 7.14 Data for exercise 7.16

Decision	μ	$\bar{\theta}^0$	$\bar{\sigma}^0$
1	120	100	30
2	110	105	30
3	100	110	30
4	90	115	30
5	80	120	30

Table 7.15 Data for exercise 7.17

7.18 In section 7.3, we showed for the transient learning problem that if M_t is the identity matrix, that the knowledge gradient for a transient truth was the same as the knowledge gradient for a stationary environment. Does this mean that the knowledge gradient produces the same behavior in both environments?

PART III - STATE-DEPENDENT PROBLEMS

We now transition to a much richer class of dynamic problems where some aspect of the *problem* depends on dynamic information. This might arise in three ways:

- The function depends on dynamic information, such as a cost or price.
- The constraints may depend on the availability of resources (that are being controlled dynamically), or other information in constraints such as the travel time in a graph or the rate at which water is evaporating.
- The distribution of a random variable such as weather, or the distribution of demand.

When we worked on state-independent problems, we often wrote the function being maximized as $F(x, W)$ to express the dependence on the decision x or random information W, but not on any information in our state S_t (or S^n). As we move to our state-dependent world, we are going to write our cost or contribution function as $C(S_t, x, W)$ to capture the possible dependence of the objective function on dynamic information in S_t. In addition, our decision x_t might be constrained by $x_t \in \mathcal{X}_t$, where the constraints $\mathcal{X}_t$ may depend on dynamic data such as inventories, travel times or conversion rates.

Finally, our random information W may itself depend on uncertain parameters. For example, W might be the number of clicks on an ad which is described by some probability distribution whose parameters (e.g. the mean) is also uncertain. Thus, at time t (or time n), we may find ourselves solving a problem that looks like

$$\max_{x_t \in \mathcal{X}_t} \mathbb{E}_{S_t} \mathbb{E}_{W|S_t} \{C(S_t, x_t, W_{t+1}) | S_t\}$$

If the cost/contribution function $C(S_t, x_t, W_{t+1})$, and/or the constraints $\mathcal{X}_t$, and/or the expectation depends on time-dependent data, then we have an instance of a state-dependent problem.

We are not trying to say that all state-dependent problems are the same, but we do claim that state-dependent problems represent an important transition from state-independent problems, where the only state is the belief B_t about our function. This is why we also refer to this class as learning problems.

We lay the foundation for state-dependent problems with the following chapters:

- State-dependent applications (chapter 8) - We begin our presentation with a series of applications of problems where the function is state dependent. State variables can arise in the objective function (e.g. prices), but in most of the applications the state arises in the constraints, which is typical of problems that involve the management of physical resources.

- Modeling general dynamic programs (chapter 9) - This chapter provides a comprehensive summary of how to model general (state-dependent) sequential decision problems in all of their glory.

- Modeling uncertainty (chapter 10) - To find good policies (to make good decisions), you need a good model, and this means an accurate model of uncertainty. In this chapter we identify different sources of uncertainty and discuss how to model them.

- Policies (chapter 11) - Here we provide a more comprehensive overview of the different strategies for creating policies, leading to the four classes of policies that we first introduce in part I for learning problems.

After these chapters, the remainder of the book is a tour through the four classes of policies which we first introduced in chapter 7. However, in chapter 7, we found that most of the interest was on two classes: parametric cost function approximations (PFAs), and direct lookaheads (DLAs). By contrast, in the richer family of state-dependent problems, we are going to need to draw on all four classes.

CHAPTER 8

STATE-DEPENDENT PROBLEMS

In chapters 5 and 7, we introduced sequential decision problems in which the state variable consisted only of the belief state about an unknown function which we represented as $\mathbb{E}\{F(x,W)|S_0\}$ where x must fall in a feasible region $\mathcal{X}$. These problems cover a very important class of applications that involve maximizing or minimizing functions that can represent anything from complex analytical functions and black-box simulators to laboratory and field experiments.

The distinguishing feature of these problems is that the problem being optimized does not depend on our state variable, where the "problem" might be the function $F(x,W)$, the expectation (e.g. the distribution of W or distributional information in S_0), or the feasible region $\mathcal{X}$. Rather, the state captures only our belief about $\mathbb{E}F(x,W)$ which affects the decision being made. For these "state-independent" problems, the (belief) state only impacts the decision through the policy.

There is a genuinely vast range of problems where the performance metric (costs or contributions), the distributions of random variables (W or distributional information in S_0) and/or the constraints, depends on the state. Examples of state variables that affect the problem itself might include:

- Physical state variables, which might include inventories, the location of a vehicle on a graph, the medical condition of a patient, the speed and location of a robot, and the condition of an aircraft engine. Physical state variables are typically expressed through the constraints.

- Informational state variables, such as prices, information about weather, or the humidity in a lab. These variables might affect the objective function (costs and contributions), or the constraints. This information may evolve exogenously, or might be directly controlled (e.g. setting the price of a product), or influenced by decisions (sell energy into the grid may lower electricity prices).

- Distributional information, which means information about the distribution of W or distributions about uncertain parameters imbedded in S_0 (such as the belief about a demand, price elasticity, or a physical property of a material).

While physical resource management problems are perhaps the easiest to envision, state dependent problems can include any problem where the function being minimized depends on dynamic information, either in the objective function itself, or the constraints. These problems typically come in one of three styles:

History independent states - Here, the state S_t reflects new information that just arrived that is not a function of either S_{t-1} or the previous decision x_{t-1}. For example, the number of customers clicking on an ad or purchasing a product may depend on the price now, but otherwise does not depend on any prior states or decisions.

Uncontrolled first-order Markov - In this case, S_t depends on S_{t-1}, but not on x_{t-1}. For example, imagine that we have a newsvendor problem where the price at which we can sell the newspapers evolves according to

$$p_t = p_{t-1} + \varepsilon_t^p,$$

where ε_t^p represents random, exogenous changes to the price. In this problem, the state $S_t = p_t$, and the decision x_t made at time t has no impact on the state S_{t+1}.

Controlled problems - Finally, we have problems where the decision x_t does impact the state S_{t+1}. For example, in a reservoir problem where R_t is the water in the reservoir, x_t is the release, and $\hat{R}_{t+1}$ is the random inflow (e.g. rainfall) between t and $t+1$, we might use the transition equation

$$R_{t+1} = R_t - x_t + \hat{R}_{t+1}.$$

In this case, we have to obey the constraint $x_t \leq R_t$, so the constraint depends on our state $S_t = R_t$.

In addition, we may encounter all of the above. The energy in a storage device represents a controlled resource, which we have to optimize in the presence of dynamically changing wind and loads (which depend on the previous time period), along with prices and variability due to sunlight (which may not depend on history).

In all of these cases, we would write our one-period cost or contribution function as $C(S_t, x_t)$ (or $C(S_t, x_t, W_{t+1})$ or $C(S_t, x_t, S_{t+1})$), rather than $F(x, W)$ as we did with our learning problems. Although we will eventually be able to exploit the simplicity of the first class of problems, for now we are going to group them together in a broad class of "state-dependent" stochastic optimization problems. While we use notation such as $C(S_t, x_t, W_{t+1})$, the state variable S_t may contain information that affects expectations, which we communicate through conditional expectations such as $\mathbb{E}\{F(x, W)|S_t\}$. Finally, we indicate the dependence on constraints using notation $x \in \mathcal{X}_t$. We could write $\mathcal{X}(S_t)$, but $\mathcal{X}_t$ is more compact and indicates that the constraints are computed at time t, and therefore may depend on S_t. We note that we cannot similarly write $C_t(x_t)$, since the t index on $C(\cdot)$ would indicate that the cost *function* depends on time, as opposed to a function that depends on data in S_t.

At this point, it is useful to highlight what is probably the biggest class of problems, which is those that involve the management of physical resources. These problems are the basis of the largest and most difficult problems that we will encounter. These problems are often high-dimensional, often with complex dynamics and types of uncertainty. Decisions generally focus on managing these physical resources, often directly (assigning a driver to a rider) or indirectly (influencing their behavior through prices).

We present three classes of examples. 1) Deterministic problems, where everything is known; 2) stochastic problems, where some information is unknown but which are described by a known probability distribution; and 3) information acquisition problems, where we have uncertainty described by an unknown distribution. In the last problem class, the focus is on collecting information so that we can better estimate the distribution.

These illustrations are designed to teach by example. The careful reader will pick up subtle modeling choices, in particular the indexing with respect to time. We suggest that readers skim these problems, selecting examples that are of interest. In chapter 9, we are going to present a very general modeling framework, and it helps to have a sense of the complexity of applications that may arise.

8.1 GRAPH PROBLEMS

A popular class of stochastic optimization problems involve managing a single physical asset moving over a graph, where the nodes of the graph capture the physical state.

8.1.1 A stochastic shortest path problem

We are often interested in shortest path problems where there is uncertainty in the cost of traversing a link. For our transportation example, it is natural to view the travel time on a link as random, reflecting the variability in traffic conditions on each link. There are two ways we can handle this uncertainty. The simplest is to assume that our driver has to make a decision before seeing the travel time over the link. In this case, our updating equation would look like

$$v_i^n = \min_{j \in \mathcal{I}_i^+} \mathbb{E}\{c_{ij}(W) + v_j^{n-1}\}$$

where W is some random variable that contains information about the network (such as travel times). This problem is identical to our original problem; all we have to do is to let $c_{ij} = \mathbb{E}\{c_{ij}(W)\}$ be the expected cost on an arc.

An alternative model is to assume that we know the travel time on a link from i to j as soon as we arrive at node i. In this case, we would have to solve

$$v_i^n = \mathbb{E}\left\{ \min_{j \in \mathcal{I}_i^+} \left(c_{ij}(W) + v_j^{n-1}\right) \right\}.$$

Here, the expectation is outside of the $\min$ operator that chooses the best decision, capturing the fact that now the decision itself is random.

Note that our notation is ambiguous, in that with the same notation, we have two very different models. In chapter 9, we are going to refine our notation so that it will be immediately apparent when a decision "sees" the random information and when the decision has to be made before the information becomes available.

8.1.2 The nomadic trucker

A nice illustration of sequential decisions is a problem we are going to call the nomadic trucker. In this problem, our trucker has to move loads of freight (which fill his truck) from one city to the next. When he arrives in a city i, he is offered a set of loads to different destinations j, and has to choose one. Once he makes his choice, he moves the load to its destination, delivers the freight and then the problem repeats itself. The other loads are offered to other drivers, so if he returns to node i at a later time, he is offered an entirely new set of loads (that are entirely random).

We model the state of our nomadic trucker by letting R_t be his location. From a location, our trucker is able to choose from a set of demands $\hat{D}_t$. Thus, our state variable is $S = (R_t, \hat{D}_t)$, where R_t is a scalar (the location) while $\hat{D}_t$ is a vector giving the number of loads from R_t to each possible destination. An action $a_t \in \mathcal{A}_t$ represents the action to accept a load in $\hat{D}_t$ and go to the destination of that load. Let $C(S_t, a_t)$ be the contribution earned from being in location R_t and taking action a_t. Any demands not covered in $\hat{D}_t$ at time t are lost. After implementing action a_t, the driver will either stay in his current location (if he does nothing), or moves to a location that corresponds to the destination of the load the driver selected in the set $\hat{D}_t$. Let R_t^a be this downstream location (which corresponds to the post-decision state variable for R_t). The post-decision state variable $S_t^a = (R_t^a)$. We assume that the action a deterministically determines the downstream destination, so $R_{t+1} = R_t^a$.

The driver makes his decision by solving

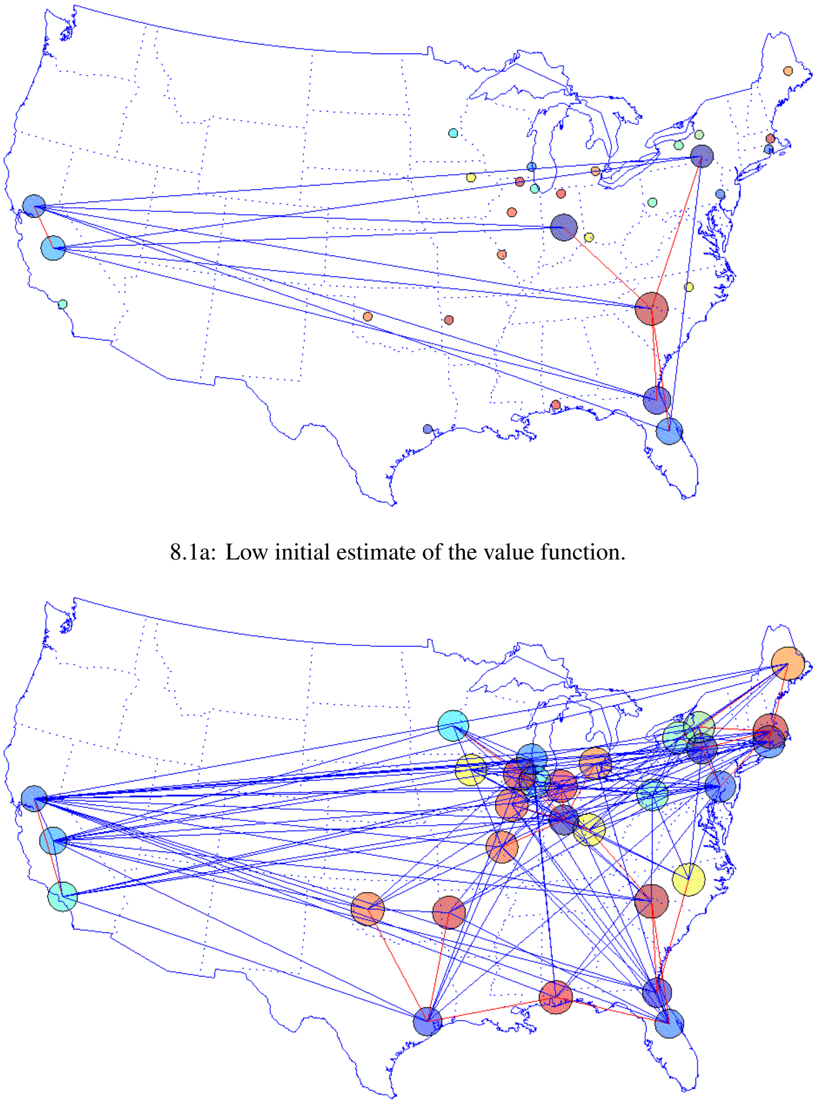
$$\hat{v}_t = \max_{a \in \hat{D}_t} \big(C(S_t, a) + \gamma \overline{V}_{t+1}(R_t^a) \big),$$

where $R_t^a = S^{M,a}(S_t, a)$ is the downstream location (in chapter 8, we will see that this is the *post-decision state*), and $\overline{V}_t(R_t^a)$ is our current estimate (as of time t) of the value of the truck being in the destination R_t^a. Let a_t be the optimal action. Noting that R_t is the current location of the truck, we update the value of our previous, post-decision state using

$$\overline{V}_{t+1}(R_t) \leftarrow (1 - \alpha_{n-1})\overline{V}_t(R_t) + \alpha_{n-1}\hat{v}_t.$$

We start by initializing the value of being in each location to zero, and use a pure exploitation strategy. If we simulate 500 iterations of this process, we produce the pattern shown in figure 8.1a. Here, the circles at each location are proportional to the value $\overline{V}^{500}(R)$ of being in that location. The small circles indicate places where the trucker never visited. Out of 30 cities, our trucker has ended up visiting seven.

This problem is an illustration of the exploration-exploitation problem, which we first saw in chapter 7. However, now we have to deal with a physical state, where what we learn depends on the physical state (the location of our trucker). In our derivative-free problems in chapter 7, we always faced the same set of possible choices at each step. Our nomadic trucker, on the other hand, faces different choices each time he moves from one city to the next. This changes the value of what he learns, since the decisions are different in each state.



8.1a: Low initial estimate of the value function.

8.1b: High initial estimate of the value function.

Figure 8.1 The effect of value function initialization on search process. Case (a) uses a low initial estimate and produces limited exploration; case (b) uses a high initial estimate, which forces exploration of the entire state space.

8.1.3 The transformer replacement problem

The electric power industry uses equipment known as transformers to convert the high-voltage electricity that comes out of power generating plants into currents with successively lower voltage, finally delivering the current we can use in our homes and businesses. The largest of these transformers can weigh 200 tons, might cost millions of dollars to replace and may require a year or more to build and deliver. Failure rates are difficult to estimate (the most powerful transformers were first installed in the 1960's and have yet to reach the end of their natural lifetime). Actual failures can be very difficult to predict, as they often depend on heat, power surges, and the level of use.

We are going to build an aggregate replacement model where we only capture the age of the transformers. Let

$$\begin{aligned} r &= \text{The age of a transformer (in units of time periods) at time } t, \\ R_{tr} &= \text{The number of active transformers of age } r \text{ at time } t. \end{aligned}$$

Here and elsewhere, we need to model the attributes of a resource (in this case, the age). While "a" might be the obvious notation, this conflicts with our notation for actions. Instead, we use "r" for the attributes of a resource, which can be a scalar or, in other applications, a vector.

For our model, we assume that age is the best predictor of the probability that a transformer will fail. Let

$$\begin{aligned} \hat{R}_{tr} &= \text{The number of transformers of age } r \text{ that fail between } t-1 \text{ and } t, \\ p_r &= \text{The probability a transformer of age } r \text{ will fail between } t-1 \text{ and } t. \end{aligned}$$

Of course, $\hat{R}_{tr}$ depends on R_{tr} since transformers can only fail if we own them.

It can take a year or two to acquire a new transformer. Assume that we are measuring time, and therefore age, in fractions of a year (say, three months). Normally it can take about six time periods from the time of purchase before a transformer is installed in the network. However, we may pay extra and get a new transformer in as little as three quarters. If we purchase a transformer that arrives in six time periods, then we might say that we have acquired a transformer that is $r = -6$ time periods old. Paying extra gets us a transformer that is $r = -3$ time periods old. Of course, the transformer is not productive until it is at least $r = 0$ time periods old. Let

$$\begin{aligned} x_{tr} &= \text{Number of transformers of age } r \text{ that we purchase at time } t, \\ c_r &= \text{The cost of purchasing a transformer of age } r. \end{aligned}$$

If we have too few transformers, then we incur what are known as "congestion costs," which represent the cost of purchasing power from more expensive utilities because of bottlenecks in the network. To capture this, let

$$\begin{aligned} \bar{R} &= \text{Target number of transformers that we should have available,} \\ R_t^A &= \text{Actual number of transformers that are available at time } t, \\ &= \sum_{r \geq 0} R_{tr}, \\ C_t(R_t^A, \bar{R}) &= \text{Expected congestion costs if } R_t^A \text{ transformers are available,} \\ &= c_0 \left(\frac{\bar{R}}{R_t^A} \right)^{\beta}. \end{aligned}$$

The function $C_t(R_t^A, \bar{R})$ captures the behavior that as R_t^A falls below $\bar{R}$, the congestion costs rise quickly.

Assume that x_{tr} is determined immediately after R_{tr} is measured. The transition function is given by

$$R_{t+1,r} = R_{t,r-1} + x_{t,r-1} - \hat{R}_{t+1,r}.$$

Let R_t, $\hat{R}_t$, and x_t be vectors with components R_{tr}, $\hat{R}_{tr}$, and x_{tr}, respectively. We can write our system dynamics more generally as

$$R_{t+1} \quad = \quad R^M(R_t, x_t, \hat{R}_{t+1}).$$

If we let $V_t(R_t)$ be the value of having a set of transformers with an age distribution described by R_t, then, as previously, we can write this value using Bellman's equation

$$V_t(R_t) = \min_{x_t} \left(cx_t + \mathbb{E}V_{t+1}(R^M(R_t, x_t, \hat{R}_{t+1})) \right).$$

For this application, our state variable R_t might have as many as 100 dimensions. If we have, say, 200 transformers, each of which might be as many as 100 years old, then the number of possible values of R_t could be 100^{200}. Fortunately, we can develop continuous approximations that allow us to approximate problems such as this relatively easily.

8.1.4 Asset valuation

Imagine you are holding an asset that you can sell at a price that fluctuates randomly. In this problem we want to determine the best time to sell the asset, and from this, infer the value of the asset. For this reason, this type of problem arises frequently in the context of asset valuation and pricing.

Let $\hat{p}_t$ be the price that is revealed in period t, at which point you have to make a decision

$$x_t = \begin{cases} 1 & \text{Sell.} \\ 0 & \text{Hold.} \end{cases}$$

For our simple model, we assume that $\hat{p}_t$ is independent of prior prices (a more typical model would assume that the *change* in price is independent of prior history). With this assumption, our system has two states:

$$S_t = \begin{cases} 1 & \text{We are holding the asset,} \\ 0 & \text{We have sold the asset.} \end{cases}$$

Assume that we measure the state immediately after the price $\hat{p}_t$ has been revealed but before we have made a decision. If we have sold the asset, then there is nothing we can do. We want to maximize the price we receive when we sell our asset. Let the scalar V_t be the value of holding the asset at time t. This can be written

$$V_t = \max_{x_t \in \{0,1\}} \left(x_t \hat{p}_t + (1 - x_t)\gamma \mathbb{E}V_{t+1} \right).$$

So, we either get the price $\hat{p}_t$ if we sell, or we get the discounted future value of the asset. Assuming the discount factor $\gamma < 1$, we do not want to hold too long simply because the value in the future is worth less than the value now. In practice, we eventually will see a price $\hat{p}_t$ that is greater than the future expected value, at which point we would stop the process and sell our asset.

The time at which we sell our asset is known as a *stopping time*. By definition, $a_\tau = 1$. It is common to think of τ as the decision variable, where we wish to solve

$$\max_\tau \mathbb{E}\hat{p}_\tau. \tag{8.1}$$

Equation (8.1) is a little tricky to interpret. Clearly, the choice of when to stop is a random variable since it depends on the price $\hat{p}_t$. We cannot optimally choose a random variable, so what is meant by (8.1) is that we wish to choose a *function* (or *policy*) that determines when we are going to sell. For example, we would expect that we might use a rule that says

$$X_t^\pi(S_t|\bar{p}) = \begin{cases} 1 & \text{if } \hat{p}_t \geq \bar{p} \text{ and } S_t = 1, \\ 0 & \text{otherwise.} \end{cases} \tag{8.2}$$

In this case, we have a function parameterized by $\bar{p}$ which allows us to write our problem in the form

$$\max_{\bar{p}} \mathbb{E} \sum_{t=1}^{\infty} \gamma^t X_t^\pi(S_t|\bar{p}).$$

This formulation raises two questions. First, while it seems very intuitive that our policy would take the form given in equation (8.2), there is the theoretical question of whether this in fact is the structure of an optimal policy. The second question is how to find the best policy within this class. For this problem, that means finding the parameter $\bar{p}$. For problems where the probability distribution of the random process driving prices is (assumed) known, this is a rich and deep theoretical challenge. Alternatively, there is a class of algorithms from stochastic optimization that allows us to find "good" values of the parameter in a fairly simple way.

8.2 INVENTORY PROBLEMS

Another popular class of problems involving managing a quantity of resources that are held in some sort of inventory. The inventory can be money, products, blood, people, water in a reservoir or energy in a battery. The decisions govern the quantity of resource moving into and out of the inventory.

8.2.1 The gambling problem

A gambler has to determine how much of his capital he should bet on each round of a game, where he will play a total of N rounds. He will win a bet with probability p and lose with probability $q = 1 - p$ (assume $q < p$). Let s^n be his total capital after n plays, $n = 1, 2, \ldots, N$, with s^0 being his initial capital. For this problem, we refer to s^n as the state of the system. Let x^n be the (discrete) amount he bets in round n, where we require that $x^n \leq s^{n-1}$. He wants to maximize $\ln s^N$ (this provides a strong penalty for ending up with a small amount of money at the end and a declining marginal value for higher amounts).

Let

$$W^n = \begin{cases} 1 & \text{if the gambler wins the } n^{th} \text{ game,} \\ 0 & \text{otherwise.} \end{cases}$$

The system evolves according to

$$S^n = S^{n-1} + x^n W^n - x^n(1 - W^n).$$

Let $V^n(S^n)$ be the value of having S^n dollars at the end of the n^{th} game. The value of being in state S^n at the end of the n^{th} round can be written as

$$\begin{aligned} V^n(S^n) &= \max_{0 \leq x^{n+1} \leq S^n} \mathbb{E}\{V^{n+1}(S^{n+1})|S^n\} \\ &= \max_{0 \leq x^{n+1} \leq S^n} \mathbb{E}\{V^{n+1}(S^n + x^{n+1}W^{n+1} - x^{n+1}(1 - W^{n+1}))|S^n\}. \end{aligned}$$

Here, we claim that the value of being in state S^n is found by choosing the decision that maximizes the expected value of being in state S^{n+1} given what we know at the end of the n^{th} round.

We solve this by starting at the end of the N^{th} trial, and assuming that we have finished with S^N dollars. The value of this is

$$V^N(S^N) = \ln S^N.$$

Now step back to $n = N - 1$, where we may write

$$\begin{aligned} V^{N-1}(S^{N-1}) &= \max_{0 \leq x^N \leq S^{N-1}} \mathbb{E}\{V^N(S^{N-1} + x^N W^N - x^N(1 - W^N))|S^{N-1}\} \\ &= \max_{0 \leq x^N \leq S^{N-1}} \left[p \ln(S^{N-1} + x^N) + (1 - p) \ln(S^{N-1} - x^N)\right]. \end{aligned} \quad (8.3)$$

Let $V^{N-1}(S^{N-1}, x^N)$ be the value within the max operator. We can find x^N by differentiating $V^{N-1}(S^{N-1}, x^N)$ with respect to x^N, giving

$$\begin{aligned} \frac{\partial V^{N-1}(S^{N-1}, x^N)}{\partial x^N} &= \frac{p}{S^{N-1} + x^N} - \frac{1 - p}{S^{N-1} - x^N} \\ &= \frac{2S^{N-1}p - S^{N-1} - x^N}{(S^{N-1})^2 - (x^N)^2}. \end{aligned}$$

Setting this equal to zero and solving for x^N gives

$$x^N = (2p - 1)S^{N-1}.$$

The next step is to plug this back into (8.3) to find $V^{N-1}(s^{N-1})$ using

$$\begin{aligned} V^{N-1}(S^{N-1}) &= p \ln(S^{N-1} + S^{N-1}(2p - 1)) + (1 - p) \ln(S^{N-1} - S^{N-1}(2p - 1)) \\ &= p \ln(S^{N-1} 2p) + (1 - p) \ln(S^{N-1} 2(1 - p)) \\ &= p \ln S^{N-1} + (1 - p) \ln S^{N-1} + \underbrace{p \ln(2p) + (1 - p) \ln(2(1 - p))}_{K} \\ &= \ln S^{N-1} + K, \end{aligned}$$

where K is a constant with respect to S^{N-1}. Since the additive constant does not change our decision, we may ignore it and use $V^{N-1}(S^{N-1}) = \ln S^{N-1}$ as our value function for $N - 1$, which is the same as our value function for N. Not surprisingly, we can keep applying this same logic backward in time and obtain

$$V^n(S^n) = \ln S^n \ (+K^N)$$

for all n, where again, K^n is some constant that can be ignored. This means that for all n, our optimal solution is

$$x^n = (2p - 1)S^{n-1}.$$

The optimal strategy at each iteration is to bet a fraction $\beta = (2p - 1)$ of our current money on hand. Of course, this requires that $p > .5$.

8.2.2 The asset acquisition problem - I

A basic asset acquisition problem arises in applications where we purchase product at time t to be used during time interval $t+1$. We are going to encounter this problem again, sometimes as discrete problems (where we would use action a), but often as continuous problems, and sometimes as vector valued problems (when we have to acquire different types of assets). For this reason, we use x as our decision variable.

We can model the problem using

R_t = The assets on hand at time t before we make a new ordering decision, and before we have satisfied any demands arising in time interval t,

x_t = The amount of product purchased at time t to be used during time interval $t+1$,

$\hat{D}_t$ = The random demands that arise between $t-1$ and t.

We have chosen to model R_t as the resources on hand in period t before demands have been satisfied. Our definition here makes it easier to introduce (in the next section) the decision of how much demand we should satisfy.

We assume we purchase new assets at a fixed price p^p and sell them at a fixed price p^s. The amount we earn between $t-1$ and t, including the decision we make at time t, is given by

$$C_t(x_t) = p^s \min\{R_t, \hat{D}_t\} - p^p x_t.$$

Our inventory R_t is described using the equation

$$R_{t+1} = R_t - \min\{R_t, \hat{D}_t\} + x_t.$$

We assume that any unsatisfied demands are lost to the system.

This problem can be solved using Bellman's equation. For this problem, R_t is our state variable. Let $V_t(R_t)$ be the value of being in state R_t. Then Bellman's equation tells us that

$$V_t(R_t) = \max_{x_t} \big(C_t(x_t) + \gamma \mathbb{E} V_{t+1}(R_{t+1}) \big).$$

where the expectation is over all the possible realizations of the demands $\hat{D}_{t+1}$.

8.2.3 The asset acquisition problem - II

Many asset acquisition problems introduce additional sources of uncertainty. The assets we are acquiring could be stocks, planes, energy commodities such as oil, consumer goods, and blood. In addition to the need to satisfy random demands (the only source of uncertainty we considered in our basic asset acquisition problem), we may also have randomness in the prices at which we buy and sell assets. We may also include exogenous changes to the assets on hand due to additions (cash deposits, blood donations, energy discoveries) and subtractions (cash withdrawals, equipment failures, theft of product).

We can model the problem using

$$\begin{aligned} x_t^p &= \text{Assets purchased (acquired) at time } t \text{ to be used during time interval } t+1, \\ x_t^s &= \text{Amount of assets sold to satisfy demands during time interval } t, \\ x_t &= (x_t^p, x_t^s), \\ R_t &= \text{Resource level at time } t \text{ before any decisions are made}, \\ D_t &= \text{Demands waiting to be served at time } t. \end{aligned}$$

Of course, we are going to require that $x_t^s \leq \min\{R_t, D_t\}$ (we cannot sell what we do not have, and we cannot sell more than the market demand). We are also going to assume that we buy and sell our assets at market prices that fluctuate over time. These are described using

$$\begin{aligned} p_t^p &= \text{Market price for purchasing assets at time } t, \\ p_t^s &= \text{Market price for selling assets at time } t, \\ p_t &= (p_t^s,\, p_t^p). \end{aligned}$$

Our system evolves according to several types of exogenous information processes that include random changes to the supplies (assets on hand), demands and prices. We model these using

$$\begin{aligned} \hat{R}_t &= \text{Exogenous changes to the assets on hand that occur during time interval } t, \\ \hat{D}_t &= \text{Demand for the resources during time interval } t, \\ \hat{p}_t^p &= \text{Change in the purchase price that occurs between } t-1 \text{ and } t, \\ \hat{p}_t^s &= \text{Change in the selling price that occurs between } t-1 \text{ and } t, \\ \hat{p}_t &= (\hat{p}_t^p, \hat{p}_t^s). \end{aligned}$$

We assume that the exogenous changes to assets, $\hat{R}_t$, occurs before we satisfy demands.

For more complex problems such as this, it is convenient to have a generic variable for exogenous information. We use the notation W_t to represent all the information that first arrives between $t-1$ and t, where for this problem, we would have

$$W_t = (\hat{R}_t, \hat{D}_t, \hat{p}_t).$$

The state of our system is described by

$$S_t = (R_t, D_t, p_t).$$

We represent the evolution of our state variable generically using

$$S_{t+1} = S^M(S_t, x_t, W_{t+1}).$$

Some communities refer to this as the "system model," hence our notation. We refer to this as the transition function. More specifically, the equations that make up our transition function would be

$$\begin{aligned} R_{t+1} &= R_t - x_t^s + x_t^p + \hat{R}_{t+1}, \\ D_{t+1} &= D_t - x_t^s + \hat{D}_{t+1}, \\ p_{t+1}^p &= p_t^p + \hat{p}_{t+1}^p, \\ p_{t+1}^s &= p_t^s + \hat{p}_{t+1}^s. \end{aligned}$$

The one-period contribution function is

$$C_t(S_t, x_t) = p_t^s x_t^s - p_t^p x_t.$$

We can find optimal decisions by solving Bellman's equation

$$V_t(S_t) = \max \big(C_t(S_t, x_t) + \gamma \mathbb{E} V_{t+1}(S_{t+1}^M(S_t, x_t, W_{t+1})) | S_t \big). \tag{8.4}$$

This problem allows us to capture a number of dimensions of the modeling of stochastic problems. This is a fairly classical problem, but we have stated it in a more general way by allowing for unsatisfied demands to be held for the future, and by allowing for random purchasing and selling prices.

8.2.4 The lagged asset acquisition problem

A variation of the basic asset acquisition problem we introduced in section 8.2.2 arises when we can purchase assets now to be used in the future. For example, a hotel might book rooms at time t for a date t' in the future. A travel agent might purchase space on a flight or a cruise line at various points in time before the trip actually happens. An airline might purchase contracts to buy fuel in the future. In all of these cases, it will generally be the case that assets purchased farther in advance are cheaper, although prices may fluctuate. For this problem, we are going to assume that selling prices are

$x_{tt'}$ = Assets purchased at time t to be used to satisfy demands that become known during time interval between $t' - 1$ and t',

x_t = $(x_{t,t+1}, x_{t,t+2}, \dots,),$
 = $(x_{tt'})_{t'>t},$

$\hat{D}_t$ = Demand for the resources that become known during time interval t,

$R_{tt'}$ = Total assets acquired on or before time t that may be used to satisfy demands that become known between $t' - 1$ and t',

R_t = $(R_{tt'})_{t' \geq t}.$

Now, R_{tt} is the resources on hand in period t that can be used to satisfy demands $\hat{D}_t$ that become known during time interval t. In this formulation, we do not allow x_{tt}, which would represent purchases on the spot market. If this were allowed, purchases at time t could be used to satisfy unsatisfied demands arising during time interval between $t-1$ and t.

The transition function is given by

$$R_{t+1,t'} = \begin{cases} \Big(R_{t,t} - \min(R_{t,t}, \hat{D}_t) \Big) + x_{t,t+1} + R_{t,t+1}, & t' = t+1, \\ R_{tt'} + x_{tt'}, & t' > t+1. \end{cases}$$

The one-period contribution function (measuring forward in time) is

$$C_t(R_t, \hat{D}_t) = p^s \min(R_{t,t}, \hat{D}_t) - \sum_{t'>t} p^p x_{tt'}.$$

We can again formulate Bellman's equation as in (8.4) to determine an optimal set of decisions. From a computational perspective, however, there is a critical difference. Now, x_t and R_t are vectors with elements $x_{tt'}$ and $R_{tt'}$, which makes it computationally impossible to enumerate all possible states (or actions).

8.2.5 The batch replenishment problem

One of the classical problems in operations research is one that we refer to here as the batch replenishment problem. To illustrate the basic problem, assume that we have a single type of resource that is consumed over time. As the reserves of the resource run low, it is necessary to replenish the resources. In many problems, there are economies of scale in this process. It is cheaper (on an average cost basis) to increase the level of resources in one jump (see examples).

■ **EXAMPLE 8.1**

A startup company has to maintain adequate reserves of operating capital to fund product development and marketing. As the cash is depleted, the finance officer has to go to the markets to raise additional capital. There are fixed costs of raising capital, so this tends to be done in batches.

■ **EXAMPLE 8.2**

An oil company maintains an aggregate level of oil reserves. As these are depleted, it will undertake exploration expeditions to identify new oil fields, which will produce jumps in the total reserves under the company's control.

To introduce the core elements, let

$\hat{D}_t$ = Demand for the resources during time interval t,

R_t = Resource level at time t,

x_t = Additional resources acquired at time t to be used during time interval $t+1$.

The transition function is given by

$$R^M_{t+1}(R_t, x_t, \hat{D}_{t+1}) = \max\{0, (R_t + x_t - \hat{D}_{t+1})\}.$$

Our one period cost function (which we wish to minimize) is given by

$$\begin{aligned} \hat{C}_{t+1}(R_t, x_t, \hat{D}_{t+1}) &= \text{Total cost of acquiring } x_t \text{ units of the resource} \\ &= c^f I_{\{x_t>0\}} + c^p x_t + c^h R^M_{t+1}(R_t, x_t, \hat{D}_{t+1}), \end{aligned}$$

where

c^f = The fixed cost of placing an order,

c^p = The unit purchase cost,

c^h = The unit holding cost.

For our purposes, $\hat{C}_{t+1}(R_t, x_t, \hat{D}_{t+1})$ could be any nonconvex function; this is a simple example of one. Since the cost function is nonconvex, it helps to order larger quantities at the same time.

Assume that we have a family of decision functions $X^\pi(R_t)$, $\pi \in \Pi$, for determining x_t. For example, we might use a decision rule such as

$$X^\pi(R_t) = \begin{cases} 0 & \text{if } R_t \geq s, \\ Q - R_t & \text{if } R_t < q. \end{cases}$$

where Q and q are specified parameters. In the language of dynamic programming, a decision rule such as $X^\pi(R_t)$ is known as a *policy* (literally, a rule for making decisions). We index policies by π, and denote the set of policies by Π. In this example, a combination (Q, q) represents a policy, and Π would represent all the possible values of Q and q.

Our goal is to solve

$$\min_{\pi \in \Pi} \mathbb{E} \left\{ \sum_{t=0}^{T} \gamma^t \hat{C}_{t+1}(R_t, X^\pi(R_t), \hat{D}_{t+1}) \right\}.$$

This means that we want to search over all possible values of Q and q to find the best performance (on average).

The basic batch replenishment problem, where R_t and x_t are scalars, is quite easy (if we know things like the distribution of demand). But there are many real problems where these are vectors because there are different types of resources. The vectors may be small (different types of fuel, raising different types of funds) or extremely large (hiring different types of people for a consulting firm or the military; maintaining spare parts inventories). Even a small number of dimensions would produce a very large problem using a discrete representation.

8.3 INFORMATION ACQUISITION PROBLEMS

Information acquisition is an important problem in many applications where we face uncertainty about the value of an action, but the only way to obtain better estimates of the value is to take the action. For example, a baseball manager may not know how well a particular player will perform at the plate. The only way to find out is to put him in the lineup and let him hit. The only way a mutual fund can learn how well a manager will perform may be to let her manage a portion of the portfolio. A pharmaceutical company does not know how the market will respond to a particular pricing strategy. The only way to learn is to offer the drug at different prices in test markets.

Information acquisition plays a particularly important role in approximate dynamic programming. Assume that a system is in state i and that a particular action might bring the system to state j. We may know the contribution of this decision, but we do not know the value of being in state j (although we may have an estimate). The only way to learn is to try making the decision and then obtain a better estimate of being in state j by actually visiting the state. This process of approximating a function, first introduced in chapter 3, is fundamental to stochastic optimization. For this reason, the information acquisition problem is of special importance.

Information acquisition problems are examples of dynamic optimization problems with belief states, along with other information that affects the function itself.

8.3.1 An information-collecting shortest path problem

Now assume that we have to choose a path through a network, but this time we face the problem that not only do we not know the actual travel time on any of the links of the network, we do not even know the mean or variance (we might be willing to assume that the probability distribution is normal). As with the two previous examples, we solve the problem repeatedly, and sometimes we want to try new paths just to collect more information.

There are two significant differences between this simple problem and the two previous problems. First, imagine that you are at a node i and you are trying to decide whether to follow the link from i to j_1 or from i to j_2. We have an estimate of the time to get from j_1 and j_2 to the final destination. These estimates may be correlated because they may share common links to the destination. Following the path from j_1 to the destination may teach us something about the time to get from j_2 to the destination (if the two paths share common links). The second difference is that making the decision to go from node i to node j changes the set of options that we face. In the bandit problem, we always faced the same set of slot machines.

Information-collecting shortest path problems arise in any information collection problem where the decision now affects not only the information you collect, but also the decisions you can make in the future. While we can solve basic bandit problems optimally, this broader problem class remains unsolved.

8.3.2 Medical decision making

Patients arrive at a doctor's office for treatment. They begin by providing a medical history, which we capture as a set of attributes $a_1, a_2, \ldots$ which includes patient characteristics (gender, age, weight), habits (smoking, diet, exercise patterns), results from a blood test, and medical history (e.g. prior diseases). Finally, the patient may have some health issue (fever, knee pain, elevated blood sugar, ...) that is the reason for the visit. This attribute vector can have hundreds of elements.

Assume that our patient is dealing with elevated blood sugar. The doctor might prescribe lifestyle changes (diet and exercise), or a form of medication (along with the dosage), where we can represent the choice as $d \in \mathcal{D}$. Let t index visits to the doctor, and let

$$x_{td} = \begin{matrix} 1 & \text{If the physicial chooses } d \in \mathcal{D}, \\ 0 & \text{Otherwise.} \end{matrix}$$

After the physical makes a decision x_t, we observe the change in blood sugar levels by $\hat{y}_{t+1}$ which we assume is learned at the next visit.

Let $U(a, x|\theta)$ be a linear model of patient attributes and medical decisions which we write using

$$U(a, x|\theta) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(a, x),$$

where $\phi_f(a, x)$ for $f \in \mathcal{F}$ represents features that we design from a combination of the patient attributes a (which are given) and the medical decision x. We believe that we can predict the patient response $\hat{y}$ using the logistic function

$$\hat{y}|\theta \sim \frac{e^{U(a,x|\theta)}}{1 + e^{U(a,x|\theta)}}. \tag{8.5}$$

Of course, we do not know what θ. We can use data across a wide range of patients to get a population estimate $\bar{\theta}^{pop}_t$ that is updated every time we treat a patient and then observe an outcome. A challenge with medical decision-making is that every patient responds to treatments differently. Ideally, we would like to estimate $\bar{\theta}_{ta}$ that depends on the attribute a of the patient.

This is a classical learning problem similar to the derivative-free problems we saw in chapter 7, with one difference: we are given the attribute a of a patient, after which we

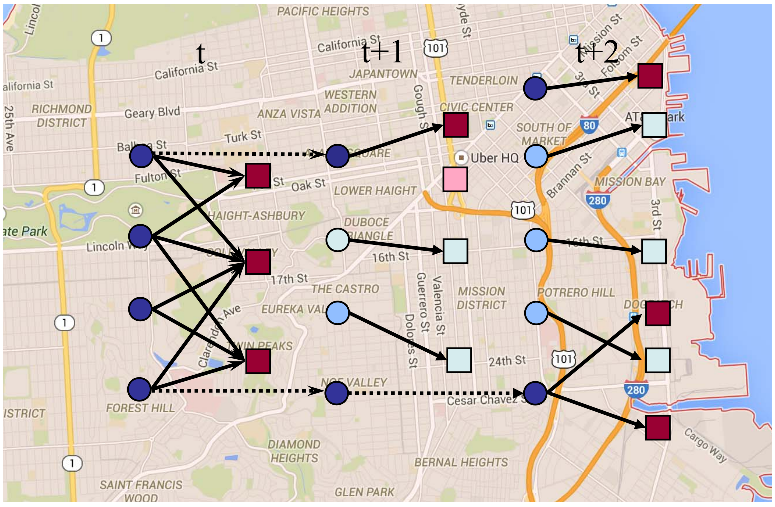


Figure 8.2 Illustration of the dynamic assignment of drivers (circles) to riders (squares).

make a decision. Our state, then, consists of our estimate of $\bar{\theta}_t^{pop}$ (or $\bar{\theta}_{ta}$), which goes into our belief state, along with the patient attribute a_t, which affects the response function itself.

8.4 COMPLEX RESOURCE ALLOCATION PROBLEMS

Problems involving the management of physical resources can become quite complex. Below we illustrate a dynamic assignment problem that arises in the context of assigning fleets of drivers (and cars) to riders requesting trips over time, and a problem involving the modeling inventories of different types of blood.

8.4.1 The dynamic assignment problem

Consider the challenge of matching drivers (or perhaps driverless electric vehicles) to customers calling in dynamically over time, illustrated in figure 8.2. We have to think about which driver to assign to which rider based on the characteristics of the driver (or car), such as where the driver lives (or how much energy is in the car's battery), along with the characteristics of the trip (origin, destination, length).

We describe drivers (and cars) using

$$a_t = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} \text{The location of the car} \\ \text{The type of car} \\ \text{How many hours the driver has been on duty} \end{pmatrix}.$$

We can model our fleet of drivers and cars using

$$\begin{aligned} R_{ta} &= \text{The number of cars with attribute } a \text{ at time } t, \\ \mathcal{R} &= \text{Set of all possible values of the attribute vector } a, \\ R_t &= (R_{ta})_{a\in\mathcal{A}}. \end{aligned}$$

We note that R_t can be very high dimensional, since the attribute a is a vector. In practice, we never generate the vector R_t, since it is more practical to just create a list of drivers and cars. The notation R_t is used just for modeling purposes.

Demands for trips arise over time, which we can model using

b = The characteristics of a trip (origin, destination, length, car type requested),

$\mathcal{B}$ = The set of all possible values of the vector b,

$\hat{D}_{tb}$ = The number of new customer requests with attribute b that were first learned at time t,

$\hat{D}_t$ = $(\hat{D}_{tb})_{b\in\mathcal{B}}$,

D_{tb} = The total number of unserved trips with attribute b waiting at time t,

D_t = $(D_{tb})_{b\in\mathcal{B}}$.

We next have to model the decisions that we have to make. Assume that at any point in time, we can either assign a technician to handle a new installation, or we can send the technician home. Let

$\mathcal{D}^H$ = The set of decisions representing sending a driver to his/her home location,

$\mathcal{D}^D$ = The set of decisions to assign a driver to a rider, where $d \in \mathcal{D}^D$ represents a decision to serve a demand of type b_d,

d^ϕ = The decision to "do nothing,"

$\mathcal{D}$ = $\mathcal{D}^H \cup \mathcal{D}^D \cup d^\phi$.

A decision has the effect of changing the attributes of a driver, as well as possibly satisfying a demand. The impact on the resource attribute vector of a driver is captured using the attribute transition function, represented using

$$a_{t+1} = a^M(a_t, d).$$

For algebraic purposes, it is useful to define the indicator function

$$\delta_{a'}(a_t, d) = \begin{cases} 1 & \text{for } a^M(a_t, d) = a', \\ 0 & \text{otherwise.} \end{cases}$$

A decision $d \in \mathcal{D}^D$ means that we are serving a customer described by an attribute vector b_d. This is only possible, of course, if $D_{tb} > 0$. Typically, D_{tb} will be 0 or 1, although our model allows for multiple trips with the same attributes. We indicate which decisions we have made using

x_{tad} = The number of times we apply a decision of type d to trip with attribute a,

x_t = $(x_{tad})_{a\in\mathcal{A},d\in\mathcal{D}}$.

Similarly, we define the cost of a decision to be

c_{tad} = The cost of applying a decision of type d to driver with attribute a,

c_t = $(c_{tad})_{a\in\mathcal{A},d\in\mathcal{D}}$.

We could solve this problem myopically by making what appears to be the best decisions now, ignoring their impact on the future. We would do this by solving

$$\min_{x_t} \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} c_{tad} x_{tad} \tag{8.6}$$

subject to

$$\begin{aligned} \sum_{d \in \mathcal{D}} x_{tad} &= R_{ta} && (8.7) \\ \sum_{a \in \mathcal{A}} x_{tad} &\leq D_{tb_d} \quad d \in \mathcal{D}^D && (8.8) \\ x_{tad} &\geq 0. && (8.9) \end{aligned}$$

Equation (8.7) says that we either have to send a technician home, or assign him to a job. Equation (8.8) says that we can only assign a technician to a job of type b_d if there is in fact a job of type b_d. Said differently, we cannot assign more than one technician per job. But we do not have to assign a technician to every job (we may not have enough technicians).

The problem posed by equations (8.6)-(8.9) is a linear program. Real problems may involve managing hundreds or even thousands of individual entities. The decision vector $x_t = (x_{tad})_{a \in \mathcal{A}, d \in \mathcal{D}}$ may have over ten thousand dimensions. But commercial linear programming packages handle problems of this size quite easily.

If we make decisions by solving (8.6)-(8.9), we say that we are using a *myopic policy* since we are using only what we know now, and we are ignoring the impact of decisions now on the future. For example, we may decide to send a technician home rather than have him sit in a hotel room waiting for a job. But this ignores the likelihood that another job may suddenly arise close to the technician's current location. Alternatively, we may have two different technicians with two different skill sets. If we only have one job, we might assign what appears to be the closest technician, ignoring the fact that this technician has specialized skills that are best reserved for difficult jobs.

Given a decision vector, the dynamics of our system can be described using

$$\begin{aligned} R_{t+1,a} &= \sum_{a' \in \mathcal{A}} \sum_{d \in \mathcal{D}} x_{ta'd} \delta_a(a', d), && (8.10) \\ D_{t+1,b_d} &= D_{t,b_d} - \sum_{a \in \mathcal{A}} x_{tad} + \hat{D}_{t+1,b_d}, \quad d \in \mathcal{D}^D. && (8.11) \end{aligned}$$

Equation (8.10) captures the effect of all decisions (including serving demands) on the attributes of the drivers. This is easiest to visualize if we assume that all tasks are completed within one time period. If this is not the case, then we simply have to augment the state vector to capture the attribute that we have partially completed a task. Equation (8.11) subtracts from the list of available demands any of type b_d that are served by a decision $d \in \mathcal{D}^D$ (recall that each element of $\mathcal{D}^D$ corresponds to a type of trip, which we denote b_d).

The state of our system is given by

$$S_t = (R_t, D_t).$$

The evolution of our state variable over time is determined by equations (8.10) and (8.10). We can now set up an optimality recursion to determine the decisions that minimize costs

	Recipient							
Donor	$AB+$	$AB-$	$A+$	$A-$	$B+$	$B-$	$O+$	$O-$
$AB+$	X							
$AB-$	X	X						
$A+$	X		X					
$A-$	X	X	X	X				
$B+$	X				X			
$B-$	X	X			X	X		
$O+$	X		X		X		X	
$O-$	X	X	X	X	X	X	X	X

Table 8.1 Allowable blood substitutions for most operations, 'X' means a substitution is allowed (from Cant (2006)).

over time using

$$V_t = \min_{x_t \in \mathcal{X}_t} \left(C_t(S_t, x_t) + \gamma \mathbb{E} V_{t+1}(S_{t+1}) \right),$$

where S_{t+1} is the state at time $t+1$ given that we are in state S_t and action x_t. S_{t+1} is random because at time t, we do not know $\hat{D}_{t+1}$. The feasible region $\mathcal{X}_t$ is defined by equations (8.7)-(8.9).

Needless to say, the state variable for this problem is quite large. The dimensionality of R_t is determined by the number of attributes of our technician, while the dimensionality of D_t is determined by the relevant attributes of a demand. In real applications, these attributes can become fairly detailed. Fortunately, the methods of approximate dynamic programming can handle these complex problems.

8.4.2 The blood management problem

The problem of managing blood inventories serves as a particularly elegant illustration of a resource allocation problem. We are going to start by assuming that we are managing inventories at a single hospital, where each week we have to decide which of our blood inventories should be used for the demands that need to be served in the upcoming week.

We have to start with a bit of background about blood. For the purposes of managing blood inventories, we care primarily about blood type and age. Although there is a vast range of differences in the blood of two individuals, for most purposes doctors focus on the eight major blood types: $A+$ (" A positive"), $A-$ ("A negative"), $B+$, $B-$, $AB+$, $AB-$, $O+$, and $O-$. While the ability to substitute different blood types can depend on the nature of the operation, for most purposes blood can be substituted according to table 8.1.

A second important characteristic of blood is its age. The storage of blood is limited to six weeks, after which it has to be discarded. Hospitals need to anticipate if they think they can use blood before it hits this limit, as it can be transferred to blood centers which monitor inventories at different hospitals within a region. It helps if a hospital can identify blood it will not need as soon as possible so that the blood can be transferred to locations that are running short.

One mechanism for extending the shelf-life of blood is to freeze it. Frozen blood can be stored up to 10 years, but it takes at least an hour to thaw, limiting its use in emergency situations or operations where the amount of blood needed is highly uncertain. In addition, once frozen blood is thawed it must be used within 24 hours.

We can model the blood problem as a heterogeneous resource allocation problem. We are going to start with a fairly basic model which can be easily extended with almost no notational changes. We begin by describing the attributes of a unit of stored blood using

$$\begin{aligned} a &= \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} \text{Blood type } (A+, A-, \ldots) \\ \text{Age (in weeks)} \end{pmatrix}, \\ \mathcal{B} &= \text{Set of all attribute types.} \end{aligned}$$

We will limit the age to the range $0 \leq a_2 \leq 6$. Blood with $a_2 = 6$ (which means blood that is already six weeks old) is no longer usable. We assume that decision epochs are made in one-week increments.

Blood inventories, and blood donations, are represented using

$$\begin{aligned} R_{ta} &= \text{Units of blood of type } a \text{ available to be assigned or held at time } t, \\ R_t &= (R_{ta})_{a \in \mathcal{A}}, \\ \hat{R}_{ta} &= \text{Number of new units of blood of type } a \text{ donated between } t-1 \text{ and} \\ & \quad\; t, \\ \hat{R}_t &= (\hat{R}_{ta})_{a \in \mathcal{A}}. \end{aligned}$$

The attributes of demand for blood are given by

$$\begin{aligned} d &= \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix} = \begin{pmatrix} \text{Blood type of patient} \\ \text{Surgery type: urgent or elective} \\ \text{Is substitution allowed?} \end{pmatrix}, \\ d^{\phi} &= \text{Decision to hold blood in inventory (``do nothing''),} \\ \mathcal{D} &= \text{Set of all demand types } d \text{ plus } d^{\phi}. \end{aligned}$$

The attribute d_3 captures the fact that there are some operations where a doctor will not allow any substitution. One example is childbirth, since infants may not be able to handle a different blood type, even if it is an allowable substitute. For our basic model, we do not allow unserved demand in one week to be held to a later week. As a result, we need only model new demands, which we accomplish with

$$\begin{aligned} \hat{D}_{td} &= \text{Units of demand with attribute } d \text{ that arose between } t-1 \text{ and } t, \\ \hat{D}_t &= (\hat{D}_{td})_{d \in \mathcal{D}}. \end{aligned}$$

We act on blood resources with decisions given by:

$$\begin{aligned} x_{tad} &= \text{Number of units of blood with attribute } a \text{ that we assign to a demand} \\ & \quad\; \text{of type } d, \\ x_t &= (x_{tad})_{a \in \mathcal{A}, d \in \mathcal{D}}. \end{aligned}$$

The feasible region $\mathcal{X}_t$ is defined by the following constraints:

$$\sum_{d \in \mathcal{D}} x_{tad} = R_{ta}, \tag{8.12}$$

$$\sum_{a \in \mathcal{A}} x_{tad} \leq \hat{D}_{td}, \quad d \in \mathcal{D}, \tag{8.13}$$

$$x_{tad} \geq 0. \tag{8.14}$$

Blood that is held simply ages one week, but we limit the age to six weeks. Blood that is assigned to satisfy a demand can be modeled as being moved to a blood-type sink, denoted, perhaps, using $a_{t,1} = \phi$ (the null blood type). The blood attribute transition function $a^M(a_t, d_t)$ is given by

$$a_{t+1} = \begin{pmatrix} a_{t+1,1} \\ a_{t+1,2} \end{pmatrix} = \begin{cases} \begin{pmatrix} a_{t,1} \\ \min\{6, a_{t,2}+1\} \end{pmatrix}, & d_t = d^\phi, \\ \begin{pmatrix} \phi \\ \text{-} \end{pmatrix}, & d_t \in \mathcal{D}. \end{cases}$$

To represent the transition function, it is useful to define

$$\begin{aligned} \delta_{a'}(a,d) &= \begin{cases} 1 & a_t^x = a' = a^M(a_t, d_t), \\ 0 & \text{otherwise,} \end{cases} \\ \Delta &= \text{Matrix with } \delta_{a'}(a,d) \text{ in row } a' \text{ and column } (a,d). \end{aligned}$$

We note that the attribute transition function is deterministic. A random element would arise, for example, if inspections of the blood resulted in blood that was less than six weeks old being judged to have expired. The resource transition function can now be written

$$\begin{aligned} R^x_{ta'} &= \sum_{a\in\mathcal{A}} \sum_{d\in\mathcal{D}} \delta_{a'}(a,d) x_{tad}, \\ R_{t+1,a'} &= R^x_{ta'} + \hat{R}_{t+1,a'}. \end{aligned}$$

In matrix form, these would be written

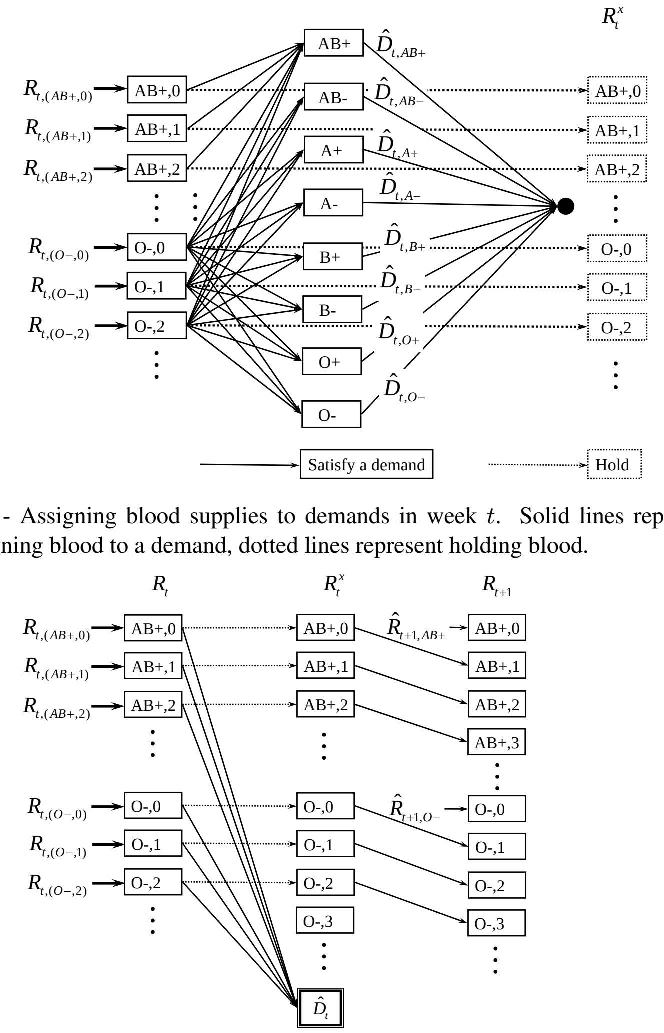
$$R^x_t = \Delta x_t, \tag{8.15}$$

$$R_{t+1} = R^x_t + \hat{R}_{t+1}. \tag{8.16}$$

Figure 8.3 illustrates the transitions that are occurring in week t. We either have to decide which type of blood to use to satisfy a demand (figure 8.3a), or to hold the blood until the following week. If we use blood to satisfy a demand, it is assumed lost from the system. If we hold the blood until the following week, it is transformed into blood that is one week older. Blood that is six weeks old may not be used to satisfy any demands, so we can view the bucket of blood that is six weeks old as a sink for unusable blood (the value of this blood would be zero). Note that blood donations are assumed to arrive with an age of 0. The pre- and post-decision state variables are given by

$$\begin{aligned} S_t &= (R_t, \hat{D}_t), \\ S^x_t &= (R^x_t). \end{aligned}$$

There is no real "cost" to assigning blood of one type to demand of another type (we are not considering steps such as spending money to encourage additional donations, or transporting inventories from one hospital to another). Instead, we use the contribution function to capture the preferences of the doctor. We would like to capture the natural preference that it is generally better not to substitute, and that satisfying an urgent demand is more important than an elective demand. For example, we might use the contributions described in table 8.2. Thus, if we use $O-$ blood to satisfy the needs for an elective patient



8.3a - Assigning blood supplies to demands in week t. Solid lines represent assigning blood to a demand, dotted lines represent holding blood.

8.3b - Holding blood supplies until week $t+1$.

Figure 8.3 The assignment of different blood types (and ages) to known demands in week t (8.3a), and holding blood until the following week (8.3b).

with $A+$ blood, we would pick up a -\$10 contribution (penalty since it is negative) for substituting blood, a +\$5 for using $O-$ blood (something the hospitals like to encourage), and a +\$20 contribution for serving an elective demand, for a total contribution of +\$15.

The total contribution (at time t) is finally given by

$$C_t(S_t, x_t) = \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} c_{tad} x_{tad}.$$

Condition	Description	Value
if $d = d^{\phi}$	Holding	0
if $a_1 = a_1$ when $d \in \mathcal{D}$	No substitution	0
if $a_1 \neq a_1$ when $d \in \mathcal{D}$	Substitution	-10
if $a_1 =$ O- when $d \in \mathcal{D}$	O- substitution	5
if $d_2 =$ Urgent	Filling urgent demand	40
if $d_2 =$ Elective	Filling elective demand	20

Table 8.2 Contributions for different types of blood and decisions

As before, let $X^{\pi}_t(S_t)$ be a policy (some sort of decision rule) that determines $x_t \in \mathcal{X}_t$ given S_t. We wish to find the best policy by solving

$$\max_{\pi \in \Pi} \mathbb{E} \sum_{t=0}^{T} C(S_t, X^{\pi}(S_t)). \tag{8.17}$$

The most obvious way to solve this problem is with a simple myopic policy, where we maximize the contribution at each point in time without regard to the effect of our decisions on the future. We can obtain a family of myopic policies by adjusting the one-period contributions. For example, our bonus of \$5 for using $O-$ blood (in table 8.2), is actually a type of myopic policy. We encourage using $O-$ blood since it is generally more available than other blood types. By changing this bonus, we obtain different types of myopic policies that we can represent by the set Π^M, where for $\pi \in \Pi^M$ our decision function would be given by

$$X^{\pi}_t(S_t) = \arg\max_{x_t \in \mathcal{X}_t} \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} c_{tad} x_{tad}. \tag{8.18}$$

The optimization problem in (8.18) is a simple linear program (known as a “transportation problem”). Solving the optimization problem given by equation (8.17) for the set $\pi \in \Pi^M$ means searching over different values of the bonus for using $O-$ blood.

8.5 BIBLIOGRAPHIC NOTES

Most of the problems in this chapter are fairly classic, in particular the deterministic and stochastic shortest path problems (see Bertsekas et al. (1991)), asset acquisition problem (see Porteus (1990), for example) and the batch replenishment problem (see **?**, among others).

Section **??** - The shortest path problem is one of the most widely studied problems in optimization. One of the early treatments of shortest paths is given in the seminal book on network flows by **?**. It has long been recognized that shortest paths could be solved directly (if inefficiently) using Bellman’s equation.

Section 8.1.1 - Many problems in discrete stochastic dynamic programming can at least conceptually be formulated as some form of stochastic shortest path problem. There is an extensive literature on stochastic shortest paths (see, for example, Frank (1969), **?**, Frieze & Grimmet (1985), Andreatta & Romeo (1988), Psaraftis & Tsitsiklis (1993), Bertsekas et al. (1991)).

Section 8.4.1 - The dynamic assignment problem is based on Powell et al. (2003).

Section 2.1.3 - Bandit problems have long been studied as classic exercises in information collection. For good introductions to this material, see Ross (1983) and Whittle (1982). A more detailed discussion of bandit problems is given in chapter 7.

PROBLEMS

8.1 What is the distinguishing characteristic of a state-dependent *problem*, as opposed to the state-independent problems we considered in chapters 5 and 7? Contrast what we mean by a solution to a stochastic optimization problem with a state-independent function, versus what we mean by a solution to a stochastic optimization problem with a state-dependent function?

8.2 Give an example of a sequential decision process from your own experience. Describe the elements of your problem following the framework provided in section 2.2. Then describe the types of rules you might use to make a decision.

8.3 Describe the gambling problem in section 8.2.1 as a decision tree, assuming that we can gamble only 0, 1 or 2 dollars in each round (this is just to keep the decision tree from growing too large).

8.4 Repeat the gambling problem assuming that the value of ending up with S^N dollars is $\sqrt{S^N}$.

8.5 Consider three variations of a shortest path problem:

Case I - All costs are known in advance. Here, we assume that we have a real-time network tracking system that allows us to see the cost on each link of the network before we start our trip. We also assume that the costs do not change during the time from which we start the trip to when we arrive at the link.

Case II - Costs are learned as the trip progresses. In this case, we assume that we see the actual link costs for links out of node i when we arrive at node i.

Case III - Costs are learned after the fact. In this setting, we only learn the cost on each link after the trip is finished.

Let v_i^I be the expected cost to get from node i to the destination for Case I. Similarly, let v_i^{II} and v_i^{III} be the expected costs for cases II and III. Show that $v_i^I \leq v_i^{II} \leq v_i^{III}$.

8.6 We are now going to do a budgeting problem where the reward function does not have any particular properties. It may have jumps, as well as being a mixture of convex and concave functions. But this time we will assume that $R = \$30$ dollars and that the allocations x_t must be in integers between 0 and 30. Assume that we have $T = 5$ products, with a contribution function $C_t(x_t) = cf(x_t)$ where $c = (c_1, \ldots, c_5) = (3, 1, 4, 2, 5)$ and where $f(x)$ is given by

$$f(x) = \begin{cases} 0, & x \leq 5, \\ 5, & x = 6, \\ 7, & x = 7, \\ 10, & x = 8, \\ 12, & x \geq 9. \end{cases}$$

Find the optimal allocation of resources over the five products.

8.7 You suddenly realize towards the end of the semester that you have three courses that have assigned a term project instead of a final exam. You quickly estimate how much each one will take to get 100 points (equivalent to an A+) on the project. You then guess that if you invest t hours in a project, which you estimated would need T hours to get 100 points, then for $t < T$ your score will be

$$R = 100\sqrt{t/T}.$$

That is, there are declining marginal returns to putting more work into a project. So, if a project is projected to take 40 hours and you only invest 10, you estimate that your score will be 50 points (100 times the square root of 10 over 40). You decide that you cannot spend more than a total of 30 hours on the projects, and you want to choose a value of t for each project that is a multiple of 5 hours. You also feel that you need to spend at least 5 hours on each project (that is, you cannot completely ignore a project). The time you estimate to get full score on each of the three projects is given by

Project	Completion time T
1	20
2	15
3	10

You decide to solve the problem as a dynamic program.

(a) What is the state variable and decision epoch for this problem?

(b) What is your reward function?

(c) Write out the problem as an optimization problem.

(d) Set up the optimality equations.

(e) Solve the optimality equations to find the right time investment strategy.

8.8 Rewrite the transition function for the asset acquisition problem II (section 8.2.3) assuming that R_t is the resources on hand after we satisfy the demands.

8.9 Write out the transition equations for the lagged asset acquisition problem in section 8.2.4 when we allow spot purchases, which means that we may have $x_{tt} > 0$. x_{tt} refers to purchases that are made at time t which can be used to serve unsatisfied demands D_t that occur during time interval t.

8.10 Identify three examples of problems where you have to try an action to learn about the reward for an action.

CHAPTER 9

MODELING DYNAMIC PROGRAMS

Perhaps one of the most important skills to develop in stochastic optimization is the ability to write down a model of the problem. Everyone who wants to solve a linear program learns to write out

$$\min_x c^T x$$

subject to

$$\begin{aligned} Ax &= b, \\ x &\geq 0. \end{aligned}$$

This standard modeling framework allows people around the world to express their problem in a standard format.

Just as popular is the format used for deterministic optimal control, where a problem might be written

$$\min_{u_0,\ldots,u_T} \sum_{t=0}^{T} L(x_t, u_t), \tag{9.1}$$

where $L(x_t, u_t)$ is a (typically nonlinear) loss function and where the state x_t evolves according to

$$x_{t+1} = f(x_t, u_t). \tag{9.2}$$

Stochastic Optimization and Learning. By Warren B. Powell

The controls may be subject to constraints.

Sequential decision problems in the presence of uncertainty is a much richer class of problems than deterministic optimization problems. For example, we have to model the sequencing of decisions, the flow of information, and potentially complex dynamics governing how the system evolves over time. These problems arise in a wide range of settings, and as a result different communities have evolved modeling styles that reflect both the nature of the problem class as well as the mathematics familiar to the host community.

The canonical models in chapter 2 illustrate these modeling styles. Our issue, and the reason for this chapter, is that none of these styles, by themselves, capture the full richness of this problem class. Yet, as you address the challenge of modeling *real* problems with software that is going to be used for real planning, the range of modeling issues can be astonishing.

Up to now, we have avoided discussing some important subtleties that arise in the modeling of stochastic, dynamic systems. We intentionally overlooked trying to define a state variable, which we have viewed as simply S_t, where the set of states might be given by the indexed set $\mathcal{S} = \{1, 2, \ldots, |\mathcal{S}|\}$. We have avoided discussions of how to properly model time or more complex information processes. We have also ignored the richness of modeling all the different sources of uncertainty for which we have a dedicated chapter (chapter 10). This style has facilitated introducing some basic ideas in dynamic programming, but would severely limit our ability to apply these methods to real problems.

The goal of this chapter is to describe a standard modeling framework for dynamic programs, providing a vocabulary that will allow us to take on a much wider set of applications (including all of the canonical models in chapter 2). Notation is not as critical for simple problems, as long as it is precise and consistent. But what seems like benign notational decisions for a simple problem can cause unnecessary difficulties, possibly making the model completely intractable as problems become more complex.

Complex problems require considerable discipline in notation because they combine the details of the original physical problem with the challenge of modeling sequential information and decision processes. The modeling of time can be particularly subtle. In addition to a desire to model problems accurately, we also need to be able to understand and exploit the structure of the problem, which can become lost in a sea of complex notation.

Section 9.1 begins by describing some basic guidelines for notational style, while section 9.2 addresses the critical question of modeling time. These two sections lay the critical foundation for notation that is used throughout the book.

The general framework for modeling a dynamic program is covered in sections 9.3 to 9.7. There are five elements to any sequential decision problem, consisting of the following:

State variables (section 9.3) - The state variables describes what we need to know (from history) to model the system forward in time. The initial state S_0 is also where we specify both known and unknown parameters.

Decision/action/control variables (section 9.4) - These are the variables we control. Choosing these variables ("making decisions") represents the central challenge in stochastic optimization. This is where we describe constraints that limit what decisions we can make. Here is where we introduce the concept of a policy, but do not describe how to design the policy.

Exogenous information processes (section 9.5) - These variables describe information that arrives to us exogenously, representing what we learn after we make each decision.

Transition function (section 9.6) - This is the function that describes how each state variable evolves from one point in time to another.

Objective function (section 9.7) - We are either trying to maximize a contribution function (profits, rewards, gains, utility) or minimize costs (or losses). This function describes how well we are doing at a point in time, and describes the problem of finding optimal policies.

We next illustrate our modeling framework with an illustration using an energy storage example, given in section 9.8.

Section 9.9 then introduces the concept of base models (which is what we describe in this chapter) and lookahead models, which represent one of our classes of policies (described in much greater detail in chapter 20). Most readers can stop reading after section 9.9 since the remainder of the chapter deals with more advanced modeling concepts.

Having laid this foundation, we transition to a series of topics that can be skipped on a first pass, but which help to expand the readers appreciation of modeling of dynamic systems. These include

Problem classification - Section 9.10 describes four fundamental problem classes differentiated based on whether we have state-independent or state-dependent problems, and whether we are working in an offline setting (maximizing the terminal reward) or an online setting (maximizing terminal reward).

Policy evaluation - Section 9.11 describes how to evaluate a policy using Monte Carlo simulation.

Advanced probabilistic modeling concepts - For readers who enjoy bridging to more advanced concepts in probability theory, section 9.12 provides an introduction to measure-theoretic concepts and probability modeling. This discussion is designed for readers who do not have any formal training in this area.

This chapter describes modeling in considerable depth, and as a result it is quite long. Sections marked with a '*' can be skipped on a first read. The section on more advanced probabilistic modeling is marked with a '**' to indicate that this is advanced material.

9.1 NOTATIONAL STYLE

Good modeling begins with good notation. The choice of notation has to balance traditional style with the needs of a particular problem class. Notation is easier to learn if it is mnemonic (the letters look like what they mean) and compact (avoiding a profusion of symbols). Notation also helps to bridge communities. Notation is a language: the simpler the language, the easier it is to understand the problem.

As a start, it is useful to adopt notational conventions to simplify the style of our presentation. For this reason, we adopt the following notational conventions:

Variables - Variables are *always* a single letter. We would never use, for example, CH for "holding cost."

Modeling time - We always use t to represent a point in time, while we use τ to represent an interval over time. When we need to represent different points in time, we might use $t, t', \bar{t}, t^{max}$, and so on.

Indexing vectors - Vectors are almost always indexed in the subscript, as in x_{ij}. Since we use discrete time models throughout, an activity at time t can be viewed as an element of a vector. When there are multiple indices, they should be ordered from outside in the general order over which they might be summed (think of the outermost index as the most detailed information). So, if x_{tij} is the flow from i to j at time t with cost c_{tij}, we might sum up the total cost using $\sum_t \sum_i \sum_j c_{tij} x_{tij}$. Dropping one or more indices creates a vector over the elements of the missing indices to the right. So, $x_t = (x_{tij})_{\forall i, \forall j}$ is the vector of all flows occurring at time t. Time, when present, is always the innermost index.

Indexing time - If we are modeling activities in discrete time, then t is an index and should be put in the subscript. So x_t would be an activity at time t, with the vector $x = (x_1, x_2, \ldots, x_t, \ldots, x_T)$ giving us all the activities over time. When modeling problems in continuous time, it is more common to write t as an argument, as in $x(t)$. x_t is notationally more compact (try writing a complex equation full of variables written as $x(t)$ instead of x_t).

Temporal indexing of functions - A common notational error is to index a function by time t when in fact the function itself does not depend on time, but depends on inputs that do depend on time. For example, imagine that we have a stochastic price process where the state $S_t = p_t$ which is the price of the asset, and x_t is how much we sell $x_t > 0$ or buy $x_t < 0$. We might want to write our contribution as

$$C_t(S_t, x)t = p_t x_t.$$

However, in this case the *function* does not depend on time t; it only depends on data $S_t = p_t$ that depends on time. So the proper way to write this would be

$$C(S_t, x) = p_t x_t.$$

Now imagine that our contribution function is given by

$$C_t(S_t, x_t) = \sum_{t'=t}^{t+H} p_{tt'} x_{tt'}.$$

Here, the *function* depends on time because the summation runs from t to $t + H$.

Flavors of variables - It is often the case that we need to indicate different flavors of variables, such as holding costs and order costs. These are always indicated as superscripts, where we might write c^h or c^{hold} as the holding cost. Note that while variables must be a single letter, superscripts may be words (although this should be used sparingly). We think of a variable like "c^h" as a single piece of notation. It is better to write c^h as the holding cost and c^p as the purchasing cost than to use h as the holding cost and p as the purchasing cost (the first approach uses a single letter c for cost, while the second approach uses up two letters - the roman alphabet is a scarce resource). Other ways of indicating flavors is hats ($\hat{x}$), bars ($\bar{x}$), tildes ($\tilde{x}$) and primes (x').

Iteration counters - We place iteration counters in the superscript, and we primarily use n as our iteration counter. So, x^n is our activity at iteration n. If we are using a descriptive superscript, we might write $x^{h,n}$ to represent x^h at iteration n. Sometimes algorithms require inner and outer iterations. In this case, we use n to index the outer iteration and m for the inner iteration.

While this will prove to be the most natural way to index iterations, there is potential for confusion where it may not be clear if the superscript n is an index (as we view it) or raising a variable to the n^{th} power. One notable exception to this convention is indexing stepsizes which we first saw in chapter 5. If we write α^n, it looks like we are raising α to the nth power, so we use α_n.

Sets are represented using capital letters in a calligraphic font, such as $\mathcal{X}, \mathcal{F}$ or $\mathcal{I}$. We generally use the lowercase roman letter as an element of a set, as in $x \in \mathcal{X}$ or $i \in \mathcal{I}$.

Exogenous information - Information that first becomes available (from outside the system) at time t is denoted using hats, for example, $\hat{D}_t$ or $\hat{p}_t$. Our only exception to this rule is W_t which is our generic notation for exogenous information (since W_t *always* refers to exogenous information, we do not use a hat).

Statistics - Statistics computed using exogenous information are generally indicated using bars, for example $\bar{x}_t$ or $\overline{V}_t$. Since these are functions of random variables, they are also random. We do not use hats, because they do not represent exogenous information; rather, they are statistics that are directly or indirectly functions of exogenous information.

Index variables - Throughout, i, j, k, l, m and n are always scalar indices.

Superscripts/subscripts on superscripts/subscripts - As a general rule, avoid superscripts on superscripts (and so forth). For example, it is tempting to think of x_{b_t} as saying that x is a function of time t, when in fact this means it is a function of b which itself depends on time.

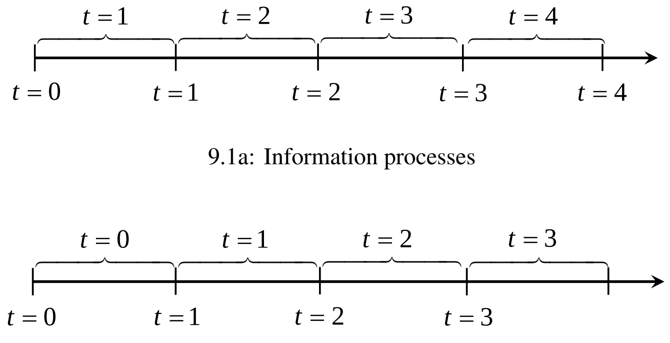
For example, x might be the number of clicks when the bid at time t is b_t, but what this notation is saying is that the number of clicks just depends on the bid, and not on time. If we want to capture the effect of both the bid and time, we have to write $x_{b,t}$.

Similarly, the notation F_{T^D} cannot be used as the forecast of the demand D at time T. To do this, you should write F^D_T. The notation F_{T^D} is just a forecast at a time $t = T^D$ that might correspond to the time, say, at which a demand occurs. But if you also write F_{T^p} where it just happens that $T^D = T^p$, you cannot refer to these as different forecasts because one is indexed by T^D while the other is indexed by T^p.

Of course, there are exceptions to every rule. It is extremely common in the transportation literature to model the flow of a type of resource (called a commodity and indexed by k) from i to j using x^k_{ij}. Following our convention, this should be written x_{kij}. It is necessary to strike a balance between a standard notational style and existing conventions.

9.2 MODELING TIME

A survey of the literature reveals different styles toward modeling time. When using discrete time, some authors start at 1 while others start at zero. When solving finite horizon



9.1a: Information processes

9.1b: Physical processes

Figure 9.1 Relationship between discrete and continuous time for information processes (9.1a) and physical processes (9.1b).

problems, it is popular to index time by the number of time periods remaining, rather than elapsed time. Some authors index a variable, say S_t, as being a function of information up through $t-1$, while others assume it includes information up through time t. The variable t may be used to represent when a physical event actually happens, or when we first know about a physical event.

The confusion over modeling time arises in part because there are two processes that we have to capture: the flow of information, and the flow of physical and financial resources. For example, a buyer may purchase an option now (an information event) to buy a commodity in the future (the physical event). Customers may call an airline (the information event) to fly on a future flight (the physical event). An electric power company has to purchase equipment now to be used one or two years in the future. All of these problems represent examples of *lagged information processes* and force us to explicitly model the informational and physical events (see section 8.2.4 for an illustration).

Notation can easily become confused when an author starts by writing down a deterministic model of a physical process, and then adds uncertainty. The problem arises because the proper convention for modeling time for information processes is different than what should be used for physical processes.

We begin by establishing the relationship between discrete and continuous time. All of the models in this book assume that decisions are made in discrete time (sometimes referred to as *decision epochs*). However, the flow of information, and many of the physical processes being modeled, are best viewed in continuous time.

The relationship of our discrete time approximation to the real flow of information and physical resources is depicted in figure 9.1. Above the line, "t" refers to a time interval while below the line, "t" refers to a point in time. When we are modeling information, time $t = 0$ is special; it represents "here and now" with the information that is available at the moment. The discrete time t refers to the time interval from $t - 1$ to t (illustrated in figure 9.1a). This means that the first new information arrives during time interval 1.

This notational style means that any variable indexed by t, say S_t or x_t, is assumed to have access to the information that arrived up to time t, which means up through time interval t. This property will dramatically simplify our notation in the future. For example, assume that f_t is our forecast of the demand for electricity. If $\hat{D}_t$ is the observed demand during time interval t, we would write our updating equation for the forecast using

$$f_{t+1} = (1 - \alpha) f_t + \alpha \hat{D}_{t+1}. \tag{9.3}$$

We refer to this form as the *informational representation*. Note that the forecast f_{t+1} is written as a function of the information that became available during time interval $(t, t+1)$, given by the demand $\hat{D}_{t+1}$.

When we are modeling a physical process, it is more natural to adopt a different convention (illustrated in figure 9.1b): discrete time t refers to the time interval between t and $t+1$. This convention arises because it is most natural in deterministic models to use time to represent when something is happening or when a resource can be used. For example, let R_t be our cash on hand that we can use during day t (implicitly, this means that we are measuring it at the beginning of the day). Let $\hat{D}_t$ be the demand for cash during the day, and let x_t represent additional cash that we have decided to add to our balance (to be used during day t). We can model our cash on hand using

$$R_{t+1} = R_t + x_t - \hat{D}_t. \tag{9.4}$$

We refer to this form as the *physical representation*. Note that the left-hand side is indexed by $t+1$, while all the quantities on the right-hand side are indexed by t.

Throughout this book, we are going to use the informational representation as indicated in equation (9.3). We first saw this in our presentation of stochastic gradients in chapter 5, when we wrote the updates from a stochastic gradient using

$$x^{n+1} = x^n + \alpha_n \nabla_x F(x^n, W^{n+1}),$$

where here we are using iteration n instead of time t.

One of the more difficult modeling challenges arises when we have to represent *lagged* information processes. For example, a customer might make an airline reservation at time t to depart at a later time t'. Alternatively, we might forecast orders at time t' using information we have at time $t < t'$. These lagged problems require that we model the information variable (at time t), as well as a physical process (what is happening at time t'). We handle these problems using two time indices, a form that we refer to as the "(t, t')" notation. For example,

$\hat{D}_{tt'}$ = The demands that first become known during time interval t to be served during time interval t'.

$f_{tt'}$ = The forecast for activities during time interval t' made using the information available up through time t.

$R_{tt'}$ = The resources on hand at time t that cannot be used until time t'.

$x_{tt'}$ = The decision to purchase futures at time t to be exercised during time interval t'.

For each variable, t indexes the information content (literally, when the variable is measured or computed), while t' represents the time at which the activity takes place. Each of these variables can be written as vectors, such as

$$\begin{aligned}
\hat{D}_t &= (\hat{D}_{tt'})_{t' \geq t}, \\
f_t &= (f_{tt'})_{t' \geq t}, \\
x_t &= (x_{tt'})_{t' \geq t}, \\
R_t &= (R_{tt'})_{t' \geq t}.
\end{aligned}$$

Note that these vectors are now written in terms of the information content. For stochastic problems, this style is the easiest and most natural. If we were modeling a deterministic problem, we would drop the first index "t" and model the entire problem in terms of the second index "t'."

Using this convention it is instructive to interpret the special case where $t = t'$. $\hat{D}_{tt}$ is simply demands that arrive during time interval t, where we first learn of them when they arrive. The forecast f^D_{tt} is the same as the actual demand $\hat{D}_{tt}$. R_{tt} represents resources that we know about during time interval t and which can be used during time interval t. Finally, x_{tt} is a decision to purchase resources to be used during time interval t given the information that arrived during time interval t. In financial circles, this is referred to as purchasing on the spot market.

9.3 THE STATES OF OUR SYSTEM

The most important quantity in any sequential decision process is the state variable. This is the set of variables that captures everything that we know, and need to know, to model our system. Without question this is the most subtle, and poorly understood, dimension of modeling sequential decision problems.

9.3.1 Defining the state variable

Surprisingly, other presentations of dynamic programming spend little time defining a state variable. Bellman's seminal text [Bellman (1957), p. 81] says "... we have a physical system characterized at any stage by a small set of parameters, the *state variables*." In a much more modern treatment, Puterman first introduces a state variable by saying [Puterman (2005), p. 18] "At each decision epoch, the system occupies a *state*." In both cases, the italics are in the original manuscript, indicating that the term "state" is being introduced. In effect, both authors are saying that given a system, the state variable will be apparent from the context.

Interestingly, different communities appear to interpret state variables in slightly different ways. We adopt an interpretation that is fairly common in the control theory community, which effectively models the state variable S_t as all the information needed to model the system from time t onward. We agree with this definition, but it does not provide much guidance in terms of actually translating real applications into a formal model. We suggest the following definitions:

Definition 9.3.1. *A* **state variable** *is:*

a) Policy-dependent version *A function of history that, combined with the exogenous information (and a policy), is necessary and sufficient to compute the cost/contribution function, the decision function (the policy), and any information required to model the evolution of information needed in the cost/contribution and decision functions.*

b) Optimization version *A function of history that is necessary and sufficient to compute the cost/contribution function, the constraints, and any information required to model the evolution of information needed in the cost/contribution function and the constraints.*

Some remarks are in order:

i) The policy-dependent definition defines the state variable in terms of the information needed to compute the core model information (cost/contribution function, and the policy (or decision function)), and any other information needed to model the evolution of the core information over time (that is, their transition functions). Note that constraints (at a point in time t) are assumed to be captured by the policy. Since the policy can be any function, it could potentially be a function that includes information that does not seem relevant to the problem, and which would never be used in an optimal policy.

ii) The optimization version defines a state variable in terms of the information needed to compute the core model information (costs/contributions and constraints), and any other information needed to model the evolution of the core information over time (their transition function). This definition limits the state variable to information needed by the optimization problem, and cannot include information that is irrelevant to the core model.

iii) Both definitions include any information that might be needed to compute the evolution of core model information, as well as information needed to model the evolution of this information over time. This includes information needed to represent the stochastic behavior, which includes distributional information needed to compute or approximate expectations.

iv) Both definitions imply that the state variable includes the information needed to compute the transition function for core model information. For example, if we model a price process using

$$p_{t+1} = \theta_0 p_t + \theta_1 p_{t-1} + \theta_2 p_{t-2} + \varepsilon^p_{t+1}, \tag{9.5}$$

then the state variable for this price process would be $S_t = (p_t, p_{t-1}, p_{t-2})$. At time t, the prices p_{t-1} and p_{t-2} are not needed to compute the cost/contribution function or constraints, but they are needed to model the evolution of p_t, which is part of the cost/contribution function.

v) The adjective "minimally dimensioned" is intended to eliminate irrelevant information. For example, with our lagged price model above, we need p_t, p_{t-1} and p_{t-2} but not p_{t-3} and earlier. However, we acknowledge that "minimally dimensioned" is not well-defined mathematically, and as such, this property is best described as aspirational: we prefer state variables with the fewest dimensions possible.

vi) If we are given a forecast $f^D_{tt'},\ t' = t+1, \ldots, T$ of demand, D_t, we assume that $f^D_{tt'} = \mathbb{E}_t D_{t'}$, where $\mathbb{E}_t$ refers to the expectation made at time t. This means that the demand D_{t+1} would be given by

$$D_{t+1} = f^D_{t,t+1} + \varepsilon_{t+1},$$

where ε_{t+1} is a mean 0 error term. Similarly, forecasts evolve over time according to

$$f^D_{t+1,t'} = f^D_{tt'} + \varepsilon_{t+1,t'},\ t' = t+1, \ldots, T,$$

where $\varepsilon_{tt'}$ for $t' = t, \ldots, T$ would generally be a correlated vector of random variables. This means that if we are given $f^D_{tt'}, t' = t, \ldots, T$, then this vector of forecasts is a part of the state variable, since all of this information is needed to model the evolution of D_t. For many problems, however, forecasts are not modeled as a dynamically evolving stochastic process; instead, they are viewed as static, which means that they are not part of the state variable (more typically, they would be *latent variables*, which means variables that we are not explicitly modeling).

vii) A byproduct of our definitions is the observation that *all* properly modeled dynamic systems are Markovian, by construction. It is surprisingly common for people to make a distinction between "Markovian" and "history-dependent" processes. For example, if our price process evolves according to equation (9.5), many would call this a history-dependent process, but consider what happens when we define

$$\bar{p}_t = \begin{pmatrix} p_t \\ p_{t-1} \\ p_{t-2} \end{pmatrix}$$

and let

$$\bar{\theta}_t = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{pmatrix}$$

which means we can write

$$p_{t+1} = \bar{\theta}^T \bar{p}_t + \varepsilon_{t+1}. \tag{9.6}$$

Here we see that $\bar{p}_t$ is a vector known at time t (who cares when the information first became known?). We would stay that equation (9.6) describes a Markov process with state $S_t = (p_t, p_{t-1}, p_{t-2})$.

This definition provides a very quick test of the validity of a state variable. If there is a piece of data in either the decision function, the transition function, or the contribution function which is not in the state variable, then we do not have a complete state variable. Similarly, if there is information in the state variable that is never needed in any of these three functions, then we can drop it and still have a valid state variable.

We use the term "necessary and sufficient" so that our state variable is as compact as possible. For example, we could argue that we need the entire history of events up to time t to model future dynamics, but in practice, this is rarely the case. As we start doing computational work, we are going to want S_t to be as compact as possible. Furthermore, there are many problems where we simply do not need to know the entire history. It might be enough to know the status of all our resources at time t (the resource variable R_t). But there are examples where this is not enough.

Assume, for example, that we need to use our history to forecast the price of a stock. Our history of prices is given by $(\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_t)$. If we use a simple exponential smoothing model, our estimate of the mean price $\bar{p}_t$ can be computed using

$$\bar{p}_t = (1-\alpha)\bar{p}_{t-1} + \alpha \hat{p}_t,$$

where α is a stepsize satisfying $0 \leq \alpha \leq 1$. With this forecasting mechanism, we do not need to retain the history of prices, but rather only the latest estimate $\bar{p}_t$. As a result, $\bar{p}_t$ is called a *sufficient statistic*, which is a statistic that captures all relevant information needed to compute any additional statistics from new information. A state variable, according to our definition, is always a sufficient statistic.

Consider what happens when we switch from exponential smoothing to an N-period moving average. Our forecast of future prices is now given by

$$\bar{p}_t = \frac{1}{N} \sum_{\tau=0}^{N-1} \hat{p}_{t-\tau}.$$

Now, we have to retain the N-period rolling set of prices $(\hat{p}_t, \hat{p}_{t-1}, \ldots, \hat{p}_{t-N+1})$ in order to compute the price estimate in the next time period. With exponential smoothing, we could write

$$S_t = \bar{p}_t.$$

If we use the moving average, our state variable would be

$$S_t = (\hat{p}_t, \hat{p}_{t-1}, \ldots, \hat{p}_{t-N+1}). \tag{9.7}$$

We make a distinction between how we model S_0 and S_t for $t > 0$:

Initial state variables S_0 The initial state is where we specify initial information, which includes deterministic parameters, initial values of any dynamically varying parameters, and probability distributions about any unknown parameters (these would be our Bayesian priors).

Dynamic state variables $S_t,\ t > 0$ We use the convention that variables in S_t for $t > 0$ are variables that are changing over time. Thus, we make a point of excluding information about deterministic parameters (for example, the acceleration of gravity or the maximum speed of an aircraft). Our model may depend on these parameters (or probability distributions), but we communicate this static information through the initial state S_0, not through the dynamic state variables S_t for $t > 0$.

Below we discuss latent variables (state variables that we *choose* to approximate as deterministic, but which really are changing stochastically over time), and unobservable state variables (which are also changing stochastically, but which we cannot observe).

9.3.2 The three states of our system

To set up our discussion, assume that we are interested in solving a relatively complex resource management problem, one that involves multiple (possibly many) different types of resources which can be modified in various ways (changing their attributes). For such a problem, it is necessary to work with three types of state variables:

The physical state R_t - This is a snapshot of the status of the physical resources we are managing and their attributes. This might include the amount of water in a reservoir, the price of a stock or the location of a sensor on a network. It could also refer to the location and speed of a robot.

The information state I_t - This encompasses any other information we need to make a decision, compute the transition or compute the objective function.

The belief (or knowledge) state B_t - The belief state is information specifying a probability distribution describing an unknown parameter. The type of distribution (e.g.

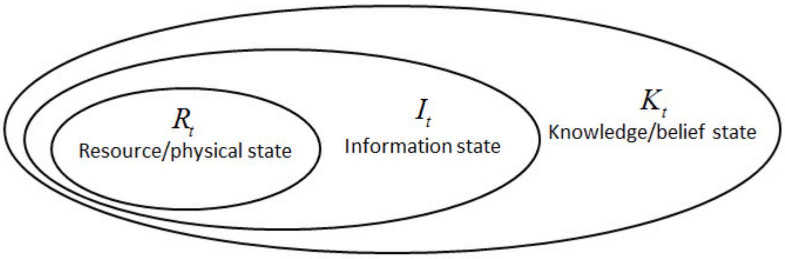


Figure 9.2 Illustration of the growing sets of state variables, where information state includes physical state variables, while the belief state includes everything.

binomial, Normal or exponential) is typically specified in the initial state S_0, although there are exceptions to this. The belief state B_t is information just like R_t and I_t, except that it is information specifying a probability distribution (such as the mean and variance of a normal distribution).

We then pull these together to create our state variable

$$S_t = (R_t, I_t, B_t).$$

Mathematically, the information state I_t should include information about resources R_t, since R_t is, after all, a form of information. The distinction between I_t (such as wind speed, temperature or the stock market), and R_t (how much energy is in the battery, water in a reservoir or money invested in the stock market) is not critical. We separate the variables simply because there are so many problems that involve managing physical or financial resources, and it is often the case that decisions impact only the physical resources. At the same time, B_t includes probabilistic information about parameters that we do not know perfectly. Knowing a parameter perfectly, as is the case with R_t and I_t, is just a special case of a probability distribution.

A proper representation of the relationship between B_t, I_t and R_t is illustrated in figure 9.2. However, we find it more useful to make a distinction (even if it is subjective) of what constitutes a variable that describes part of the physical state R_t, and then let I_t be all remaining variables that describe quantities that are known perfectly. Then, we let B_t consist entirely of probability distributions that describe parameters that we do not know perfectly.

State variables take on different flavors depending on the mixture of physical, informational and knowledge states, as well as the relationship between the state of the system now, and the states in the past.

- Physical state - There are three important variations that involve a physical state:
  - Pure physical state - There are many problems which involve only a physical state which is typically some sort of resource being managed. There are problems where R_t is a vector, a low-dimensional vector (as in $R_t = (R_{ti})_{i \in \mathcal{I}}$ where i might be a blood type, or a type of piece of equipment), or a high-dimensional vector (as in $R_t = (R_{ta})_{a \in \mathcal{A}}$ where a is a multidimenisonal attribute vector).

- Physical state with information - We may be managing the water in a reservoir (captured by R_t) given temperature and wind speed (which affects evaporation) captured by I_t.
- Physical state, information state, and belief state - We need the cash on hand in a mutual fund, R_t, information I_t about interest rates, and a probability model B_t describing, say, our belief about whether the stock market is going up or down.

- Information state - In most applications information evolves exogenously, although there are exceptions. The evolution of information comes in several flavors:
  - Memoryless - The information I_{t+1} does not depend on I_t. For example, we may feel that the characteristics of a patient arriving at time $t + 1$ to a doctor's office is independent of the patient arriving at time t. We may also believe that rainfall in month $t + 1$ is independent of the rainfall in month t.
  - First-order Markov - Here we assume that I_{t+1} depends on I_t. For example, we may feel that the spot market price of oil, the wind speed, or temperature and humidity at $t + 1$ depend on the value at time t. We might also insist that a decision x_{t+1} not deviate more than a certain amount from the decision x_t at time t.
  - Higher-order Markov - We may feel that the price of a stock p_{t+1} depends on p_t, p_{t-1}, and p_{t-2}. However, we can create a variable $\bar{p}_t = (p_t, p_{t-1}, p_{t-2})$ and convert such a system to a first-order Markov system, so we really only have to deal with memoryless and first-order Markov systems.
  - Full history dependent - This arises when the evolution of the information I_{t+1} depends on the full history, as might happen when modeling the progress of currency prices or the progression of a disease. This type of model is typically used when we are not comfortable with a compact state variable.

- Belief state - We remember what we learned after making a decision. There are many settings where the belief state is updated recursively, which means that B_{t+1} depends on B_t (this might be the mean and variance of a distribution, or a set of probabilities). Uncertainty in the belief state can arise in three ways:
  - Uncertainty about a static parameter - For example, we may not know the impact of price on demand, or the sales of a laptop with specific features. The nature of the unknown parameter depends on the type of belief model: the features of the laptop correspond to a lookup table, while the demand-price tradeoff represents the parameter of a parametric model. These problems are broadly known under the umbrella of optimal learning, but are often associated with the literature on multiarmed bandit problems.
  - Uncertainty about a dynamic (uncontrollable) parameter - The sales of a laptop with a specific set of features may change over time. This may occur because of unobservable variables. For example, the demand elasticity of a product (such as housing) may depend on other market characteristics (such as the growth of industry in the area).
  - Uncertainty about a dynamic, controllable parameter - Imagine that we control the inventory of a product that we cannot observe perfectly. We may control

purchases that replenish inventory which is then used to complete sales, but our ability to track sales is imperfect, giving us an imprecise estimate of the inventory. These problems are typically referred to as partially observable Markov decision processes (POMDPs).

xxx

There has been a tendency in the literature to treat the belief state as if it were somehow different than "the" state variable. It is not. The state variable is all the information that describes the system at time t, whether that information is the amount of inventory, the location of a vehicle, the current weather or interest rates, or the parameters of a distribution describing some unknown quantity.

We can use S_t to be the state of a single resource (if this is all we are managing), or let $S_t = R_t$ be the state of all the resources we are managing. There are many problems where the state of the system consists only of R_t. We suggest using S_t as a generic state variable when it is not important to be specific, but it must be used when we may wish to include other forms of information. For example, we might be managing resources (consumer products, equipment, people) to serve customer demands $\hat{D}_t$ that become known at time t. If R_t describes the state of the resources we are managing, our state variable would consist of $S_t = (R_t, \hat{D}_t)$, where $\hat{D}_t$ represents additional information we need to solve the problem.

9.3.3 The initial state S_0

The initial state plays a special role in the modeling of a stochastic optimization problem. By convention, S_t only contains the information that changes over time. Thus, if we were solving a shortest path problem over a deterministic graph, S_t would tell us the node which we currently occupy, but would not include, for example, the deterministic data describing the graph. Similarly, it would not include any deterministic parameters such as the maximum speed of our vehicle.

By contrast, we can use S_0 to store any data that is an input to our system. This can include

- Any deterministic parameters - This might include the deterministic data describing a graph (for example), or any problem parameters that never change.

- Initial values of parameters that evolve over time - For example, this could be the initial inventory, the starting location of a robot, or the initial speed of wind at a wind farm.

- The distribution of belief about uncertain parameters - This is known as the *prior* distribution of belief about

As our system evolves, we drop any deterministic parameters that do not change. These become *latent* (or hidden) variables, since our problem depends on them, but we drop them from S_t for $t > 0$. However, it is important to recognize that these values may change each time we solve an instance of the problem. Examples of these random starting states include:

■ **EXAMPLE 9.1**

We wish to optimize the management of a fleet of trucks. We fix the number of trucks in our fleet, but this is a parameter that we specify, and we may change the fleet size from one instance of the problem to another.

■ **EXAMPLE 9.2**

We wish to optimize the amount of energy to store in a battery given a forecast of clouds over a 24-hour planning horizon. Let $f_{0t'}$ is the forecast of energy at time t' which is given to us at time 0, the vector of forecasts $f_0 = (f_{0t'})_{t'=0}^{24}$ (which does not evolve over time) is part of the initial state. However, each time we optimize our problem, we are given a new forecast.

■ **EXAMPLE 9.3**

We are designing an optimal policy for finding the best medication for type II diabetes, but the policy depends on the attributes of the patient (age, weight, gendor, ethnicity, and medical history), which do not change over the course of the treatment.

9.3.4 The post-decision state variable

We can view our system as evolving through sequences of new information followed by a decision followed by new information (and so on). Although we have not yet discussed decisions, for the moment let the decisions (which may be a vector) be represented generically using x_t (we discuss our choice of notation for a decision in the next section). In this case, a history of the process might be represented using

$$h_t = (S_0, x_0, W_1, x_1, W_2, x_2, \dots, x_{t-1}, W_t).$$

Here, h_t contains all the information we need to make a decision x_t at time t. As we discussed before, h_t is sufficient but not necessary. We expect our state variable to capture what is needed to make a decision, allowing us to represent the history as

$$h_t = (S_0, x_0, W_1, S_1, x_1, W_2, S_2, x_2, \dots, x_{t-1}, W_t, S_t). \tag{9.8}$$

The sequence in equation (9.8) defines our state variable as occurring after new information arrives and before a decision is made. For this reason, we call S_t the *pre-decision state variable*. This is the most natural place to write a state variable because the point of capturing information from the past is to make a decision.

For most problem classes, we can design more effective computational strategies using the *post-decision state variable*. This is the state of the system after a decision x_t, which means we can drop any information needed only to make the decision. For this reason, we denote this state variable S_t^x, which produces the history

$$h_t = (S_0, x_0, S_0^x, W_1, S_1, x_1, S_1^x, W_2, S_2, x_2, S_2^x, \dots, x_{t-1}, S_{t-1}^x, W_t, S_t). \tag{9.9}$$

We again emphasize that our notation S_t^x means that this function has access to all the exogenous information up through time t, along with the decision x_t (which also has access to the information up through time t).

The examples below provide some illustrations of pre- and post-decision states.

■ EXAMPLE 9.1

A traveler is driving through a network, where the travel time on each link of the network is random. As she arrives at node i, she is allowed to see the travel times on each of the links out of node i, which we represent by $\hat{\tau}_i = (\hat{\tau}_{ij})_j$. As she arrives at node i, her pre-decision state is $S_t = (i, \hat{\tau}_i)$. Assume she decides to move from i to k. Her post-decision state is $S_t^x = (k)$ (note that she is still at node i; the post-decision state captures the fact that she will next be at node k, and we no longer have to include the travel times on the links out of node i).

■ EXAMPLE 9.2

The nomadic trucker revisited. Let $R_{ta} = 1$ if the trucker has attribute vector a at time t and 0 otherwise. Now let D_{tb} be the number of customer demands (loads of freight) of type b available to be moved at time t. The pre-decision state variable for the trucker is $S_t = (R_t, D_t)$, which tells us the state of the trucker and the demands available to be moved. Assume that once the trucker makes a decision, all the unserved demands in D_t are lost, and new demands become available at time $t+1$. The post-decision state variable is given by $S_t^x = R_t^x$ where $R_{ta}^x = 1$ if the trucker has attribute vector r after a decision has been made.

■ EXAMPLE 9.3

Imagine playing backgammon where R_{ti} is the number of your pieces on the i^{th} "point" on the backgammon board (there are 24 points on a board). The transition from S_t to S_{t+1} depends on the player's decision x_t, the play of the opposing player, and the next roll of the dice. The post-decision state variable is simply the state of the board after a player moves but before his opponent has moved.

The post-decision state can be particularly valuable in the context of dynamic programming, which we are going to address in depth in chapters 17 and 18. We can see this when we set up Bellman's equation, which we first saw in chapter 2, section 2.1.9, which is written

$$\begin{aligned} V_t(S_t) &= \max_x \big(C(S_t, x) + \mathbb{E}\{V_{t+1}(S_{t+1})|S_t\}\big), \\ &= \max_x \big(C(S_t, x) + \sum_{s'} p(s'|S_t, x) V_{t+1}(s')\big). \end{aligned}$$

Computing the expectation, which typically means finding the one-step transition matrix $p(s'|S_t, x)$, can be computationally expensive (or completely intractable). If we use the post-decision state S_t^x, we break Bellman's equation into two steps: the deterministic transition from the pre-decision state S_t to the post-decision state S_t^x, and then the stochastic transition from the post-decision state S_t^x to the next pre-decision state S_{t+1}. These can be

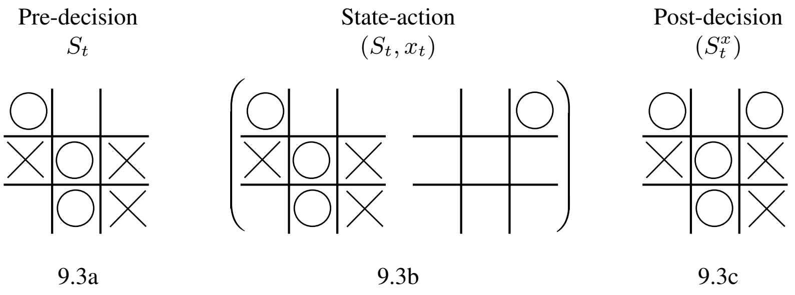


Figure 9.3 Pre-decision state, augmented state-action, and post-decision state for tic-tac-toe.

written

$$
\begin{aligned}
V_t(S_t) &= \max_x \big(C(S_t, x) + V_t^x(S_t^x)\big), && (9.10)\\
V_t^x(S_t^x) &= \mathbb{E}\{V_{t+1}(S_{t+1})|S_t^x\}. && (9.11)
\end{aligned}
$$

Equation (9.10) is attractive because we can now compute the maximum without having to then compute the imbedded expectation. In chapter 19, we are going to exploit this structure to solve problems where the action a is actually a very high-dimensional vector x. Then, equation (9.11) might be approximated using simulation-based methods.

There are three ways of finding a post-decision state variable:

Decomposing decisions and information There are many problems where we can create functions $S^{M,x}(\cdot)$ and $S^{M,W}(\cdot)$ from which we can compute

$$
\begin{aligned}
S_t^x &= S^{M,x}(S_t, x_t), && (9.12)\\
S_{t+1} &= S^{M,W}(S_t^x, W_{t+1}). && (9.13)
\end{aligned}
$$

The structure of these functions is highly problem-dependent. However, there are sometimes significant computational benefits, primarily when we face the problem of approximating the value function. Recall that the state variable captures all the information we need to make a decision, compute the transition function, and compute the contribution function. S_t^x *only* has to carry the information needed to compute the transition function. For some applications, S_t^x has the same dimensionality as S_t, but in many settings, S_t^x is dramatically simpler than S_t, simplifying the problem of approximating the value function.

State-action pairs A very generic way of representing a post-decision state is to simply write

$$
S_t^x = (S_t, x_t).
$$

Figure 9.3 provides a nice illustration using our tic-tac-toe example. Figure 9.3a shows a tic-tac-toe board just before player O makes his move. Figure 9.3b shows the augmented state-action pair, where the decision (O decides to place his move in the upper right hand corner) is distinct from the state. Finally, figure 9.3c shows the post-decision state. For this example, the pre- and post-decision state spaces are the same, while the augmented state-action pair is nine times larger.

The augmented state (S_t, x_t) is closely related to the post-decision state S_t^x (not surprising, since we can compute S_t^x deterministically from S_t and x_t). But computationally, the difference is significant. If $\mathcal{S}$ is the set of possible values of S_t, and $\mathcal{X}$ is the set of possible values of x_t, then our augmented state space has size $|\mathcal{S}| \times |\mathcal{X}|$, which is obviously much larger.

The augmented state variable is used in a popular class of algorithms known as Q-learning (which we first introduced in chapter 2), where the challenge is to statistically estimate Q-factors which give the value of being in state S_t *and* taking action x_t. The Q-factors are written $Q(S_t, x_t)$, in contrast with value functions $V_t(S_t)$ which provide the value of being in a state. This allows us to directly find the best action by solving $\min_x Q(S_t, x_t)$. This is the essence of Q-learning, but the price of this algorithmic step is that we have to estimate $Q(S_t, x_t)$ for each S_t and x_t. It is not possible to determine x_t by optimizing a function of S_t^x alone, since we generally cannot determine which action x_t brought us to S_t^x.

The post-decision as a point estimate Assume that we have a problem where we can compute a point estimate of future information. Let $\overline{W}_{t,t+1}$ be a point estimate, computed at time t, of the outcome of W_{t+1}. If W_{t+1} is a numerical quantity, we might use $\overline{W}_{t,t+1} = \mathbb{E}(W_{t+1}|S_t)$ or $\overline{W}_{t,t+1} = 0$.

If we can create a reasonable estimate $\overline{W}_{t,t+1}$, we can compute post- and pre-decision state variables using

$$\begin{aligned} S_t^x &= S^M(S_t, x_t, \overline{W}_{t,t+1}), \\ S_{t+1} &= S^M(S_t, x_t, W_{t+1}). \end{aligned}$$

Measured this way, we can think of S_t^x as a point estimate of S_{t+1}, but this does not mean that S_t^x is necessarily an approximation of the expected value of S_{t+1}.

9.3.5 Partially observable states*

There are many applications where we are not able to observe (or measure) the state of the system precisely. Some examples include:

■ **EXAMPLE 9.1**

A retailer may have to order inventory without being able to measure the precise current inventory. It is possible to measure sales, but theft and breakage introduce errors, which means we do not know the inventory exactly.

■ **EXAMPLE 9.2**

A patient may have cancer in the colon which might be indicated by the presence of polyps (small growths in the colon). The number of polyps is not directly observable. There are different methods for testing for the presence of polyps that allow us to infer how many there may be, but these are imperfect.

■ **EXAMPLE 9.3**

The military has to make decisions about sending out aircraft to remove important military targets that may have been damaged in previous raids. These decisions typically have to be made without knowing the precise state of the targets.

■ **EXAMPLE 9.4**

Policy makers have to decide how much to reduce CO2 emissions, and would like to plan a policy over 200 years that strikes a balance between costs and the rise in global temperatures. Scientists cannot measure temperatures perfectly (in large part because of natural variations), and the impact of CO2 on temperature is unknown and not directly observable.

For each of these examples, we have a system with variables (parameters) that cannot be observed, along with variables that can be observed. There are two ways to handle the unobservable varaibles:

1) We focus on modeling the variables that describe the system (both observable and unobservable), and then work to estimate the probability that the unobservable variables take on a particular value. From this perspective, the problem is referred to as a *partially observable Markov decision process* or POMDP.

2) We model the observable system, which means the observable state variables, and the beliefs about the unobservable variables. This perspective is sometimes referred to as the *belief MDP*.

9.3.5.1 Partially observable MDP

Let

$$\begin{aligned} S_t^U &= \text{The state of the unobservable system at time } t, \\ B(s) &= \text{The probability that } S_t^U = s. \end{aligned}$$

We now proceed by building our model around the unobservable state, and then develop algorithms for finding the belief distribution $B(s)$. Assume that states are discrete, where $s \in \mathcal{S}^U = \{s_1, \ldots, s_N\}$. Then our belief distribution has to satisfy

$$\sum_{s \in \mathcal{S}^U} B(s) = 1.$$

9.3.5.2 Belief MDP

Now imagine that we are trying to treat diabetes with a drug $x \in \mathcal{X} = \{x_1, \ldots, X_M\}$. Assume that μ_x is the expected reduction in blood sugar. We do not know μ_x, so we treat it as a random variable, and further assume that it is normally distributed where

$$\mu_x \sim N(\bar{\mu}_x^0, (\sigma_x^0)^2).$$

We would say that our initial belief about μ_x is normally distributed with initial mean $\bar{\mu}_x^0$ and variable $(\sigma_x^0)^2$. We can then write our belief state $B^0 = (\bar{\mu}_x^0, (\sigma_x^0)^2)$. If we have any variables that can be observed perfectly, we would use our variables R^0 for information that seems like a physical (or resource) state (such as the location of a vehicle), or I^0 for other information. In this case, our initial state would be

$$S^0 = (R^0, I^0, B^0).$$

If we do not have any parameters that are perfectly observable, then we would just write

$$S^0 = (B^0).$$

Either way, we are modeling the unobservable part of the system through the state variable B^0. In contrast to the POMDP perspective where we model the dynamics of the unobservable variables, now we model the dynamics of the belief state.

This perspective is the one we used in chapter 7 for derivative-free stochastic optimization, where these are often referred to as multi-armed bandit problems (we prefer "active learning problems").

9.3.5.3 POMDP vs. Belief MDP? There is a long history of handling unobservable systems through the framework of POMDPs, where we first model the basic system that can be controlled but not observed, and then work to find the probability that the unobservable system is in a particular state. This is typically modeled using discrete (or discretized) states, building on tools that we present later (in chapter 14). The problem is that these tools are limited to problems where the number of discrete states is not too large, which tends to limit us to very small problems.

Our approach will be to model what we can observe, even if it includes the belief distribution around unobservable elements of the system. We will not require that our state variable be continuous, or even low-dimensional.

9.3.6 Latent variables*

One of the more subtle dimensions of any dynamic model is the presence of information that is not explicitly captured in the state variable S_t. Remember that we do not model in the dynamic state S_t any information in S_0 that does not change over time. We may have many static parameters in S_0. While these are used in the model, they are not in S_t, which means that we will not explicitly capture the dependence of the model on these parameters.

For example, imagine that we are moving energy into and out of the battery, where we buy and sell a quantity x_t at a price p_t. Let R_t be the amount of energy in the battery, and let η be a parameter that governs the maximum rate at which we can move energy. Our state variable would be $S_t = (R_t, p_t)$, but would not include η, because it is a static parameter.

We are going to need to compute different functions that depend on the state, such as the value of being in a state or the policy which is the decision given the state. If we write these functions as $F(S_t)$, we have to recognize that these functions are estimated given η. Although we could write our function $F(S_t|\eta)$, we are not actually estimating a function that depends on η. This means that if we change η, we are not going to know how $F(S_t)$ changes. This means that η is being treated as a latent (that is hidden) variable.

There are many problems where there is data that changes over time, but where we ignore this change within our model. For example, forecasts of energy from a wind farm might be updated every hour, but we may want to model energy storage over a 24 hour period using a fixed set of forecasts. This means that the forecasts are being treated as latent variables.

- Travel times on a transportation network - Planning a path from an origin to a destination over a transportation network, we might assume that we know all the times on each link in the network. This is a form of deterministic dynamic program, but as we progress, the travel times will be updated. When we solve the deterministic version of the network, we are treating travel times as latent variables.

- Demand elasticity with respect to price for airline service - Imagine that we are trying to optimize revenue by optimizing the price of an airline set as a function of the number of days until the flight departs. We can solve the optimization problem assuming the elasticities are known, although we still have to deal with the uncertainty in the actual number of bookings. But as we observe the actual demand response, we may update our estimate of the elasticities. Elasticities would be viewed as latent variables in the original optimization problem.

- Battery storage capacity - We may need to solve a stochastic optimization problem governing when to charge or discharge a battery, given its power and capacity. However, the power and capacity degrade over time, requiring that we update our solution. When we optimize the policy given assumption about power and capacity, we are treating these parameters as latent variables.

In a nutshell, a latent variable is a variable that we are treating as a static parameter, but which is actually changing over time.

When deciding whether to model a variable (such as a forecast) explicitly (which means modeling how it evolves over time) or as a latent variable (which means holding it constant) introduces an important tradeoff: including a dynamically varying parameter in the state variable produces a more complex, higher dimensional state variable, but one which does not have to be reoptimized when the parameter changes. By contrast, treating a parameter as a latent variable simplifies the model, but requires that the model be reoptimized when the parameter changes.

9.3.7 Forecasts and the transition kernel*

An important dimension of any dynamic model is the probability distribution of random activities that will happen in the future. For example, imagine that we are planning the commitment of energy resources given a forecast of the energy $W_{t'}$ that will be generated from wind at some time t' in the future. Imagine that at time t, we are provided with a set of forecasts $f^W_{tt'}$ of the wind $W_{t'}$ at time t'. We might now assume that the wind in the future is given by

$$W_{t'} = f^W_{tt'} + \varepsilon_{tt'},$$

where $\varepsilon_{tt'} \approx N(0, (t'-t)\sigma^W)$.

For this simple model, where the variance of the error depends only on how far into the future we are projecting, the forecast determines the probability distribution of the energy from wind. In the vast majority of models, we treat the forecast as a latent variable, which means that if we are solving a problem over some interval $(t, t+H)$, we treat the forecast $(f^W_{tt'})^{t+H}_{t'=t}$ as fixed. In practice, the forecast evolves over time as new information arrives. We might model this evolution using

$$f^W_{t+1,t'} = f^W_{tt'} + \hat{f}^W_{t+1,t'},$$

where $\hat{f}^W_{t+1,t'}$ represents the exogenous change in the forecast for time t'.

We can introduce the dimension that the forecast itself evolves over time. This means that the vector of forecasts $(f^W_{tt'})^{t+H}_{t'=t}$ become part of the state variable. However, the forecast is simply a parameter characterizing the probability distribution governing the dynamics of our system, so this means that the probability distribution that guides the transition function (known as the transition kernel) has to be included in the state variable if it evolves (stochastically) over time.

Classical dynamic programming models seem to almost universally ignore the role of forecasts in the modeling of a dynamic optimization problem. When the forecast (or the transition probability distribution) is treated as a latent variable, then it means that the problem has to be re-optimized from scratch when the forecasts are updated. By contrast, forecasts are easily handled in lookahead policies, as we see later.

9.3.8 Flat vs. factored state representations*

It is very common in the dynamic programming literature to define a discrete set of states $\mathcal{S} = (1, 2, \ldots, |\mathcal{S}|)$, where $s \in \mathcal{S}$ indexes a particular state. For example, consider an inventory problem where S_t is the number of items we have in inventory (where S_t is a scalar). Here, our state space $\mathcal{S}$ is the set of integers, and $s \in \mathcal{S}$ tells us how many products are in inventory.

Now assume that we are managing a set of K product types. The state of our system might be given by $S_t = (S_{t1}, S_{t2}, \ldots, S_{tk}, \ldots)$ where S_{tk} is the number of items of type k in inventory at time t. Assume that $S_{tk} \leq M$. Our state space $\mathcal{S}$ would consist of all possible values of S_t, which could be as large as K^M. A state $s \in \mathcal{S}$ corresponds to a particular vector of quantities $(S_{tk})_{k=1}^K$.

Modeling each state with a single scalar index is known as a flat or unstructured representation. Such a representation is simple and elegant, and produces very compact models that have been popular in the operations research community. We first saw this used in section 2.1.9, and we will return to this in chapter 14 in much more depth. However, the use of a single index completely disguises the structure of the state variable, and often produces intractably large state spaces.

In the design of algorithms, it is often essential that we exploit the structure of a state variable. For this reason, we generally find it necessary to use what is known as a *factored* representation, where each factor represents a *feature* of the state variable. For example, in our inventory example we have K factors (or features). It is possible to build approximations that exploit the structure that each dimension of the state variable is a particular quantity.

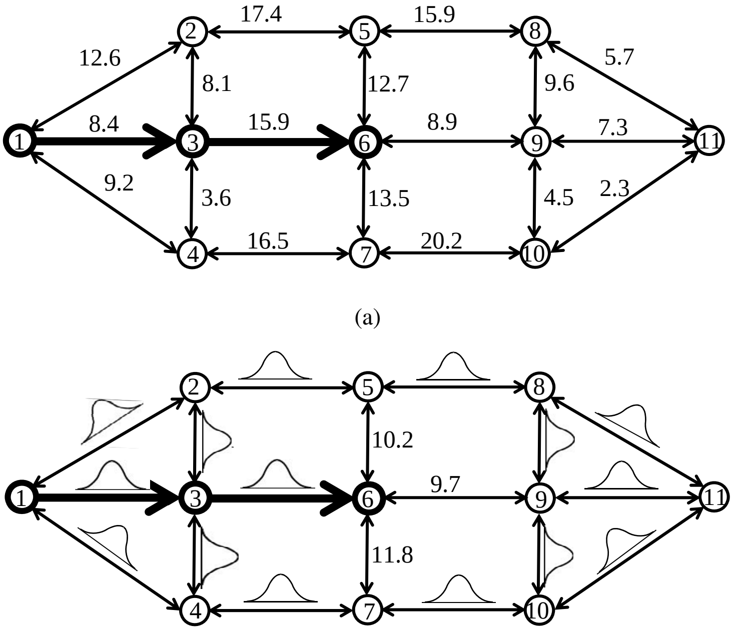
Our attribute vector notation, which we use to describe a single entity, is an example of a factored representation. Each element a_i of an attribute vector represents a particular feature of the entity. The resource state variable $R_t = (R_{ta})_{a \in \mathcal{A}}$ is also a factored representation, since we explicitly capture the number of resources with a particular attribute. This is useful when we begin developing approximations for problems such as the dynamic assignment problem that we introduced in section 8.4.1.

9.3.9 A shortest path illustration

We are going to use a simple shortest path problem to illustrate the process of defining a state variable. We start with a deterministic graph shown in figure 9.4(a), where we are interested in finding the best path from node 1 to node 11. Let t be the number of links we have traversed, and let N_t be the node number were we are located after $t = 2$ transitions. What state are we in?

Most people answer this with

$$S_t = N_t = 6.$$



(a)

(b)

Figure 9.4 (a) Deterministic network for a traveler moving from node 1 to node 11 with known arc costs. (b) Stochastic network, where arc costs are revealed as the traveler arrives to a node.

This answer hints at two conventions that we use when defining a state variable. First, we exclude any information that is not changing, which in this case is any information about our deterministic graph. It also excludes the prior nodes in our path (1 and 3) since these are not needed for any future decisions.

Now assume that the travel times are random, but where we know the probability distribution of travel times over each link (and these distributions are not changing over time). This graph is depicted in Figure 9.4(b). We are going to assume, however, that when a traveler arrives at node i, she is able to see the actual cost $\hat{c}_{ij}$ for the link (i, j) out of node i (if this is the link that is chosen now). Now, what is our state variable?

Obviously, we still need to know our current node $N_t = 6$. However, the revealed link costs also matter. If the cost of 9.6 is actually 2.3, or 18.4, our decision may change. Thus, these costs are very much a part of our state of information. Thus, we would write our state as

$$S_t = (N_t, (\hat{c}_{N_t,\cdot})) = (6, (10.2, 9.7, 11.8)),$$

where $(\hat{c}_{N_t,\cdot})$ represents the costs on all the links out of node N_t. Thus, we see an illustration of both a physical state $R_t = N_t$, and information $I_t = (10.2, 9.7, 11.8)$.

For our last example, we introduce the problem of left-hand turn penalties. If our turn from node 6 to node 5 is a left hand turn, we are going to add a penalty of .7 minutes. Now what is our state variable?

The left-hand turn penalty requires that we know if the move from 6 to 5 is a left hand turn. This calculation requires knowing where we are coming from. Thus, we now need to include our previous node, N_{t-1} in our state variable, giving us

$$S_t = (N_t, (\hat{c}_{N_t,\cdot}), N_{t-1}) = (6, (10.2, 9.7, 11.8), 3).$$

Now, N_t is our physical state, but N_{t-1} is a piece of information required to compute the cost function.

9.4 MODELING DECISIONS

Fundamental to dynamic programs is the characteristic that we are making decisions over time, In the setting of stochastic optimization, we can think of decisions as a form of information that we control, in contrast with the forms of uncertainty that we do not control. Here we discuss types of decisions, and introduce the concept of a policy which is our function for making decisions.

9.4.1 Decisions, actions, and controls

A survey of the literature reveals a distressing variety of words used to mean "decisions." The classical literature on Markov decision process talks about choosing an action $a \in \mathcal{A}$ (or $a \in \mathcal{A}_s$, where $\mathcal{A}_s$ is the set of actions available when we are in state s) or a policy (a rule for choosing an action). The optimal control community chooses a control $u \in \mathcal{U}_x$ when the system is in state x. The math programming community wants to choose a decision represented by the vector x, and the simulation community wants to apply a rule. We have also noticed that the bandit community in computer science has also adopted "x" as its notation for a decision which is typically discrete.

In our presentation, we use "x" as our default notation.

Decisions come in many forms. We illustrate this using our notation x which tends to be the notation of choice for more complex problems. Examples of different types of decisions are

- Binary, where x can be 0 or 1.
- Discrete set, where $x \in \{x_1, \ldots, x_M\}$.
- Continuous scalar, where $x \in [a, b]$.
- Continuous vector, where $x \in \Re^n$.
- Vector integer, where $x \in \mathbb{Z}$.
- Subset selection, where x is a vector of 0's and 1's, indicating which members are in the set.
- Multidimensional categorical, where $x_a = 1$ if we make a choice described by an attribute $a = (a_1, \ldots, a_K)$. For example, a could be the attributes of a drug or patient, or the features of a movie.

There are many applications where a decision is either continuous or vector-valued. For example, in chapter 8 we describe applications where a decision at time t involves the assignment of resources to tasks. Let $x = (x_d)_{d \in \mathcal{D}}$ be the vector of decisions, where $d \in \mathcal{D}$ is a *type* of decision, such as assigning resource i to task j, or purchasing a particular type of equipment. It is not hard to create problems with hundreds, thousands and even tens of thousands of *dimensions*. These high-dimensional decision vectors arise frequently in the types of resource allocation problems addressed in operations research.

This discussion makes it clear that the complexity of the space of decisions (or actions or controls) can vary considerably across applications. There are entire communities dedicated to problems with a specific class of decisions. For example, optimal stopping problems feature binary actions (hold or sell). The entire field of Markov decision processes, as well as all the problems described in chapter 7 for derivative-free stochastic optimization, assume discrete sets. Derivative-based stochastic optimization, as well as the field of stochastic programming, assumes that x is a vector, possibly with some (or all) variables being integer.

9.4.2 Using policies to make decisions

The challenge of any optimization problem (including stochastic optimization) is making decisions. In a deterministic setting, we can pose the problem as one of making a decision x, or a sequence of decisions $x_0, x_1, \ldots, x_T$, to minimize or maximize some objective. In a sequential (stochastic) decision problem, the decision x_t depends on the information available at time t, which is captured by S_t. This means we need a decision x_t for each S_t. This relationship is known as a *policy*, often designated by π. While many authors use $\pi(S_t)$ to represent the policy, we use π to carry the information that describes the function, and designate the function as $X^\pi(S_t)$. If we are using action a_t, we would designate our policy as $A^\pi(S_t)$, or $U^\pi(S_t)$ if we are finding control u_t. Policies may be stationary (as we have written them), or time-dependent, in which case we would write $X_t^\pi(S_t)$.

The heart of the field of stochastic optimization is finding effective policies. In chapter 5, we saw that we could present our gradient-based algorithm as a form of policy. In chapter 7, we introduced a number of policies for guiding our decision of what decision to test next. We also hinted at different ways of creating policies.

Starting in chapter 11, we are going to spend the rest of the book identifying different classes of policies that are suited to problems with different characteristics. This chapter can be viewed as describing how to model these systems so that we can assess how well a policy performs.

9.5 THE EXOGENOUS INFORMATION PROCESS

An important dimension of many of the problems that we address is the arrival of exogenous information, which changes the state of our system. Modeling the flow of exogenous information represents, along with states, the most subtle dimension of modeling a stochastic optimization problem. We sketch the basic notation for modeling exogenous information here, and defer to chapter 10 a more complete discussion of uncertainty.

We begin by noting that this section only addresses the exogenous information that arrives at times $t > 0$. This ignores the initial state S_0 which is an entirely different source of information (which technically is exogenous).

Sample path	$t = 0$	$t = 1$		$t = 2$		$t = 3$	
ω	p_0	$\hat{p}_1$	p_1	$\hat{p}_2$	p_2	$\hat{p}_3$	p_3
1	29.80	2.44	32.24	1.71	33.95	-1.65	32.30
2	29.80	-1.96	27.84	0.47	28.30	1.88	30.18
3	29.80	-1.05	28.75	-0.77	27.98	1.64	29.61
4	29.80	2.35	32.15	1.43	33.58	-0.71	32.87
5	29.80	0.50	30.30	-0.56	29.74	-0.73	29.01
6	29.80	-1.82	27.98	-0.78	27.20	0.29	27.48
7	29.80	-1.63	28.17	0.00	28.17	-1.99	26.18
8	29.80	-0.47	29.33	-1.02	28.31	-1.44	26.87
9	29.80	-0.24	29.56	2.25	31.81	1.48	33.29
10	29.80	-2.45	27.35	2.06	29.41	-0.62	28.80

Table 9.1 A set of sample realizations of prices (p_t) and the changes in prices ($\hat{p}_t$)

9.5.1 Basic notation for information processes

Consider a problem of tracking the value of an asset. Assume the price evolves according to

$$p_{t+1} = p_t + \hat{p}_{t+1}.$$

Here, $\hat{p}_{t+1}$ is an exogenous random variable representing the change in the price during time interval $t+1$. At time t, p_t is a number, while (at time t) p_{t+1} is random.

We might assume that $\hat{p}_{t+1}$ comes from some probability distribution such as a normal distribution with mean 0 and variance σ^2. However, rather than work with a random variable described by some probability distribution, we are going to primarily work with sample realizations. Table 9.1 shows 10 sample realizations of a price process that starts with $p_0 = 29.80$ but then evolves according to the sample realization.

Following standard convention, we index each path by the Greek letter ω (in the example below, ω runs from 1 to 10). At time $t = 0$, p_t and $\hat{p}_t$ is a random variable (for $t \geq 1$), while $p_t(\omega)$ and $\hat{p}_t(\omega)$ are *sample realizations*. We refer to the sequence

$$p_1(\omega),\ p_2(\omega),\ p_3(\omega),\ \ldots, p_T(\omega)$$

as a *sample path* for the prices p_t.

We are going to use "ω" notation throughout this volume, so it is important to understand what it means. As a rule, we will primarily index exogenous random variables such as $\hat{p}_t$ using ω, as in $\hat{p}_t(\omega)$. $\hat{p}_{t'}$ is a random variable if we are sitting at a point in time $t < t'$. $\hat{p}_t(\omega)$ is not a random variable; it is a sample realization. For example, if $\omega = 5$ and $t = 2$, then $\hat{p}_t(\omega) = -0.73$. We are going to create randomness by choosing ω at random. To make this more specific, we need to define

$$\begin{aligned}
\Omega &= \text{the set of all possible sample realizations (with } \omega \in \Omega),\\
p(\omega) &= \text{the probability that outcome } \omega \text{ will occur.}
\end{aligned}$$

A word of caution is needed here. We will often work with continuous random variables, in which case we have to think of ω as being continuous. In this case, we cannot say $p(\omega)$ is the "probability of outcome ω." However, in all of our work, we will use discrete samples. For this purpose, we can define

$$\hat{\Omega} \quad = \quad \text{A set of discrete sample observations of } \omega \in \Omega.$$

In this case, we can talk about $p(\omega)$ being the probability that we sample ω from within the set $\hat{\Omega}$.

For more complex problems, we may have an entire family of random variables. In such cases, it is useful to have a generic "information variable" that represents all the information that arrives during time interval t. For this purpose, we define

$$W_t = \text{The exogenous information becoming available during interval } t.$$

We might also say that W_t is the information that first becomes known by time t.

W_t may be a single variable, or a collection of variables (travel times, equipment failures, customer demands). We note that while we use the convention of putting hats on variables representing exogenous information ($\hat{D}_t, \hat{p}_t$), we do not use a hat for W_t since this is our only use for this variable, whereas D_t and p_t have other meanings. We always think of information as arriving in continuous time, hence W_t is the information arriving during time interval t, rather than at time t. This eliminates the ambiguity over the information available when we make a decision at time t.

The choice of notation W_t as a generic "information function" is not standard, but it is mnemonic (it looks like ω_t). We would then write $\omega_t = W_t(\omega)$ as a sample realization of the information arriving during time interval t. This notation adds a certain elegance when we need to write decision functions and information in the same equation.

Some authors use ω to index a particular sample path, where $W_t(\omega)$ is the information that arrives during time interval t. Other authors view ω as the information itself, as in

$$\omega = (-0.24, 2.25, 1.48).$$

Obviously, both are equivalent. Sometimes it is convenient to define

$$\begin{aligned} \omega_t &= \text{The information that arrives during time period } t \\ &= W_t(\omega), \\ \omega &= (\omega_1, \omega_2, \ldots). \end{aligned}$$

We sometimes need to refer to the *history* of our process, for which we define

$$\begin{aligned} H_t &= \text{The history of the process, consisting of all the information known} \\ & \text{through time } t, \\ &= (W_1, W_2, \ldots, W_t), \\ \mathcal{H}_t &= \text{The set of all possible histories through time } t, \\ &= \{H_t(\omega) | \omega \in \Omega\}, \\ h_t &= \text{A sample realization of a history,} \\ &= H_t(\omega), \\ \Omega(h_t) &= \{\omega \in \Omega | H_t(\omega) = h_t\}, \\ &= \text{The set of all sample paths that correspond to history } h_t. \end{aligned}$$

In some applications, we might refer to h_t as the state of our system, but this is usually a very clumsy representation. However, we will use the history of the process for a specific modeling and algorithmic strategy.

9.5.2 Outcomes and scenarios

Some communities prefer to use the term *scenario* to refer to a sample realization of random information. For most purposes, “outcome,” “sample path,” and “scenario” can be used interchangeably (although sample path refers to a sequence of outcomes over time). The term scenario causes problems of both notation and interpretation. First, “scenario” and “state” create an immediate competition for the interpretation of the letter “s.” Second, “scenario” is often used in the context of major events. For example, we can talk about the scenario that the Chinese might revalue their currency. We could talk about two scenarios: (1) the Chinese hold the current relationship between the yuan and the dollar, and (2) they allow their currency to float. For each scenario, we could talk about the fluctuations in the exchange rates between all currencies.

Recognizing that different communities use “outcome” and “scenario” to mean the same thing, we suggest that we may want to reserve the ability to use both terms simultaneously. For example, we might have a set of scenarios that determine if and when the Chinese revalue their currency (but this would be a small set). We recommend denoting the set of scenarios by Ψ, with $\psi \in \Psi$ representing an individual scenario. Then, for a particular scenario ψ, we might have a set of outcomes $\omega \in \Omega$ (or $\Omega(\psi)$) representing various minor events (currency exchange rates, for example).

■ **EXAMPLE 9.1**

Planning spare transformers - In the electric power sector, a certain type of transformer was invented in the 1960’s. As of this writing, the industry does not really know the failure rate curve for these units (is their lifetime roughly 50 years? 60 years?). Let ψ be the scenario that the failure curve has a particular shape (for example, where failures begin happening at a higher rate around 50 years). For a given scenario (failure rate curve), ω represents a sample realization of failures (transformers can fail at any time, although the likelihood they will fail depends on ψ).

■ **EXAMPLE 9.2**

Energy resource planning - The federal government has to determine energy sources that will replace fossil fuels. As research takes place, there are random improvements in various technologies. However, the planning of future energy technologies depends on the success of specific options, notably whether we will be able to sequester carbon underground. If this succeeds, we will be able to take advantage of vast stores of coal in the United States. Otherwise, we have to focus on options such as hydrogen and nuclear.

In chapter 20, describe a class of policies based on approximating the future using a sample, where each outcome in the sample is typically referred to as a “scenario.”

9.5.3 Lagged information processes*

There are many settings where the information about a new arrival comes before the new arrival itself as illustrated in the examples.

■ **EXAMPLE 9.1**

A customer may make a reservation at time t to be served at time t'.

■ **EXAMPLE 9.2**

An orange juice products company may purchase futures for frozen concentrated orange juice at time t that can be exercised at time t'.

■ **EXAMPLE 9.3**

A programmer may start working on a piece of coding at time t with the expectation that it will be finished at time t'.

We first saw lagged problems in section 9.2 when we introduced the (t, t') notation. Let $D_{tt'}$ be the number of customers calling in at time t to book a hotel room at time t', and let $f^p_{tt'}$ be the forecast of the price of frozen orange juice concentrate in year t' based on what we know at time t. Finally, we might make a decision $x_{tt'}$ at time t to reserve a car at time t'. In all these instances, t is the time at which the variable is being determined, and t' is simply an attribute. We can write our set of orders arriving on day t as

$$D_t = (D_{tt'})_{t' \geq t}.$$

Then, $D_1, D_2, \ldots, D_t, \ldots$ is the sequence of orders, where each D_t can be orders being called in for different times into the future.

9.5.4 Models of information processes

Information processes come in varying degrees of complexity. Needless to say, the structure of the information process plays a major role in the models and algorithms used to solve the problem. Below, we describe information processes in increasing levels of complexity.

State-independent processes

A large number of problems involve processes that evolve independently of the state of the system, such as wind (in an energy application), stock prices (in the context of small trading decisions) and demand for a product (assuming inventories are small relative to the market).

■ **EXAMPLE 9.1**

A publicly traded index fund has a price process that can be described (in discrete time) as $p_{t+1} = p_t + \sigma\delta$, where δ is normally distributed with mean μ, variance 1, and σ is the standard deviation of the change over the length of the time interval.

■ **EXAMPLE 9.2**

Requests for credit card confirmations arrive according to a Poisson process with rate λ. This means that the number of arrivals during a period of length Δt is given by a Poisson distribution with mean $\lambda\Delta t$, which is independent of the history of the system.

The practical challenge we typically face in these applications is that we do not know the parameters of the system. In our price process, the price may be trending upward or downward, as determined by the parameter μ. In our customer arrival process, we need to know the rate λ (which can also be a function of time).

State-independent information processes are attractive because they can be generated and stored in advance, simplifying the process of testing policies. In chapter 20, we will describe an algorithmic strategy based on the use of *scenario trees* which have to be created in advance.

State/action-dependent information processes

There are many problems where the exogenous information W_{t+1} depends on the state S_t and/or the decision x_t. Some illustrations include:

■ **EXAMPLE 9.1**

Customers arrive at an automated teller machine according to a Poisson process, but as the line grows longer, an increasing proportion decline to join the queue (a property known as *balking* in the queueing literature). The apparent arrival rate at the queue is a process that depends on the length of the queue.

■ **EXAMPLE 9.2**

A market with limited information may respond to price changes. If the price drops over the course of a day, the market may interpret the change as a downward movement, increasing sales and putting further downward pressure on the price. Conversely, upward movement may be interpreted as a signal that people are buying the stock, encouraging more buying behavior.

■ **EXAMPLE 9.3**

The change in the speed of wind at a wind farm depends on the current speed. If the current speed is low, the change is likely to be an increase. If it is high, the change is likely to be a decrease.

Exogenous information can also depend on the action. Imagine that a mutual fund is trying to optimize the process of selling a large position in a company. If the mutual fund makes the decision to sell a large number of shares, the effect may be to depress the stock price because the act of selling sends a negative signal to the market. Thus, the action may influence what would normally be an exogenous stochastic process.

State/action-dependent information processes make it impossible to pre-generate sample outcomes when testing policies. While not a major issue, it complicates comparing policies since we cannot fix the sample outcomes.

State-dependent information processes introduce a subtle notational complication. Following standard convention, the notation ω almost universally refers to a sample path. Thus, $W_t(\omega)$ represents the exogenous information arriving between $t-1$ and t when we are following sample path ω. If we write $S_t(\omega)$, we mean the state we are in at time t when we are following sample path ω, but now we have to make it clear what policy we are following to get there. For example, we might write $S^\pi_{t+1} = S^M(S^\pi_t, X^\pi_t(S_t), W^\pi_{t+1}(\omega))$, where it is clear that we are using policy π to get from S^π_t to S^π_{t+1}.

Adversarial behavior*

We typically model sequential stochastic decision processes as if the exogenous information process, which may be state (and action) dependent, is being drawn from an otherwise unintelligent distribution with no malicious intent. However, there are many applications where the exogenous information may come from an adversary such as a game opponent or an attacker, who is making decisions (the exogenous information) to make our system work as poorly as possible.

The behavior of an adversarial exogenous source requires its own model to reflect the objective of the adversary, but the most common approach is to replace the standard statement of Bellman's equation,

$$V_t(S_t) \quad = \quad \max_{x_t} \mathbb{E}\{C_t(S_t, x_t) + \gamma V_{t+1}(S_{t+1}) | S_t\}.$$

We can formulate this problem using a "$\min$-$\max$" formulation

$$V_t(S_t) \quad = \quad \max_{x_t} \min_{w} \big(C_t(S_t, x_t) + \gamma V_{t+1}(S_{t+1})\big)$$

where $S_{t+1} = S^M(S_t, x_t, w)$. Instead of taking an expectation over all possible outcomes, we seek the outcome w that produces the worst performance (in this case, the one that minimizes the contribution which we are trying to maximize).

More complex information processes*

Now consider the problem of modeling currency exchange rates. The change in the exchange rate between one pair of currencies is usually followed quickly by changes in others. If the Japanese yen rises relative to the U.S. dollar, it is likely that the Euro will also rise relative to it, although not necessarily proportionally. As a result, we have a vector of information processes that are correlated.

In addition to correlations between information processes, we can also have correlations over time. An upward push in the exchange rate between two currencies in one day is likely to be followed by similar changes for several days while the market responds to new information. Sometimes the changes reflect long term problems in the economy of a country. Such processes may be modeled using advanced statistical models which capture correlations between processes as well as over time.

An *information model* can be thought of as a probability density function $\phi_t(\omega_t)$ that gives the density (we would say the probability of ω if it were discrete) of an outcome ω_t in time t. If the problem has independent increments, we would write the density simply as $\phi_t(\omega_t)$. If the information process is Markovian (dependent on a state variable), then we would write it as $\phi_t(\omega_t | S_{t-1})$.

In some cases with complex information models, it is possible to proceed without any model at all. Instead, we can use realizations drawn from history. For example, we may take samples of changes in exchange rates from different periods in history and assume that these are representative of changes that may happen in the future. The value of using samples from history is that they capture all of the properties of the real system. This is an example of planning a system without a model of an information process.

9.5.5 Supervisory processes*

We are sometimes trying to control systems where we have access to a set of decisions from an exogenous source. These may be decisions from history, or they may come from a knowledgeable expert. Either way, this produces a dataset of states $(S^m)_{m=1}^n$ and decisions $(x^m)_{m=1}^n$. In some cases, we can use this information to fit a statistical model which we use to try to predict the decision that would have been made given a state.

The nature of such a statistical model depends very much on the context, as illustrated in the examples.

■ **EXAMPLE 9.1**

Consider our nomadic trucker where we measure his state s^n (his state) and his decision a^n which we represent in terms of the destination of his next move. We could use a historical file $(s^m, a^m)_{m=1}^n$ to build a probability distribution $\rho(s,a)$ which gives the probability that we make decision a given his state s. We can use $\rho(s,a)$ to predict decisions in the future.

■ **EXAMPLE 9.2**

A mutual fund manager adds x_t dollars in cash at the end of day t (to be used to cover withdrawals on day $t+1$) when there are R_t dollars in cash left over at the end of the day. We can use a series of observations of x_{t_m} and R_{t_m} on days $t_1, t_2, \ldots, t_m$ to fit a model of the form $X(R) = \theta_0 + \theta_1 R + \theta_2 R^2 + \theta_3 R^3$.

We can use supervisory processes to statistically estimate a decision function that forms an initial policy. We can then use this policy in the context of an approximate dynamic programming algorithm to help fit value functions that can be used to improve the decision function. The supervisory process helps provide an initial policy that may not be perfect, but at least is reasonable.

9.5.6 Policies in the information process*

The sequence of information $(\omega_1, \omega_2, \ldots, \omega_t)$ is assumed to be driven by some sort of exogenous process. However, we are generally interested in quantities that are functions of both exogenous information as well as the decisions. It is useful to think of decisions as *endogenous information*. But where do the decisions come from? We now see that decisions come from policies. In fact, it is useful to represent our sequence of information and decisions as

$$H_t^\pi = (S_0, X_0^\pi, W_1, S_1, X_1^\pi, W_2, S_2, X_2^\pi, \ldots, X_{t-1}^\pi, W_t, S_t). \tag{9.14}$$

Now our history is characterized by a family of functions: the information variables W_t, the decision functions (policies) X_t^π, and the state variables S_t. We see that to characterize a particular history h_t, we have to specify both the sample outcome ω as well as the policy π. Thus, we might write a sample realization as

$$h_t^\pi = H_t^\pi(\omega).$$

We can think of a complete history $H_\infty^\pi(\omega)$ as an outcome in an expanded probability space (if we have a finite horizon, we would denote this by $H_T^\pi(\omega)$). Let

$$\omega^\pi = H_\infty^\pi(\omega)$$

be an outcome in our expanded space, where ω^π is determined by ω and the policy π. Let Ω^π be the set of all outcomes of this expanded space. The probability of an outcome in Ω^π obviously depends on the policy we are following. Thus, computing expectations (for example, expected costs or rewards) requires knowing the policy as well as the set of exogenous outcomes. For this reason, if we are interested, say, in the expected costs during time period t, some authors will write $E_t^\pi\{C_t(S_t, x_t)\}$ to express the dependence of the expectation on the policy. However, even if we do not explicitly index the policy, it is important to understand that we need to know how we are making decisions if we are going to compute expectations or other quantities.

9.6 THE TRANSITION FUNCTION

The next step in modeling a dynamic system is the specification of the *transition function*. This function describes how the system evolves from one state to another as a result of decisions and information. We begin our discussion of system dynamics by introducing some general mathematical notation. While useful, this generic notation does not provide much guidance into how specific problems should be modeled. We then describe how to model the dynamics of some simple problems, followed by a more general model for complex resources.

9.6.1 A general model

The dynamics of our system are represented by a function that describes how the state evolves as new information arrives and decisions are made. The dynamics of a system can be represented in different ways. The easiest is through a simple function that works as follows

$$S_{t+1} = S^M(S_t, X_t^\pi(S_t), W_{t+1}). \tag{9.15}$$

The function $S^M(\cdot)$ goes by different names such as "plant model" (literally, the model of a physical production plant), "plant equation," "law of motion," "transfer function," "system dynamics," "system model," "state equations," "transition law," and "transition function." We prefer "transition function" because it is the most descriptive. We use the notation $S^M(\cdot)$ to reflect that this is the *state* transition function, which represents a *model* of the dynamics of the system. Below, we reinforce the "M" superscript with other modeling devices.

The arguments of the function follow standard notational conventions in the control literature (state, action, information), but different authors will follow one of two conventions for modeling time. While equation (9.15) is fairly common, the stochastic controls community will write

$$S_{t+1} = S^M(S_t, x_t, w_t), \tag{9.16}$$

where w_t is random at time t.

This is a very general way of representing the dynamics of a system. In many problems, the information W_{t+1} arriving during time interval $t+1$ depends on the state S_t at the end of time interval t, but is conditionally independent of all prior history given S_t. For example, a driver moving over a road network may only learn about the travel times on a link from i to j when he arrives at node i. When this is the case, we say that we have a Markov information process. When the decisions depend only on the state S_t, then we have a Markov decision process. In this case, we can store the system dynamics in the form of a one-step transition matrix using

$P(s'|s,x)$ = The probability that $S_{t+1} = s'$ given $S_t = s$ and $X^\pi_t = x$,

P^π = Matrix of elements where $P(s'|s,x)$ is the element in row s and column s' and where the decision x to be made in each state is determined by a policy π.

There is a simple relationship between the transition function and the one-step transition matrix. Let

$$\mathbb{1}_X = \begin{cases} 1 & X \text{ is true} \\ 0 & \text{Otherwise.} \end{cases}$$

Assuming that the set of outcomes Ω is discrete, the one-step transition matrix can be computed using

$$\begin{aligned} P(s'|s,x) &= \mathbb{E}\{\mathbb{1}_{\{s'=S^M(S_t,x,W_{t+1})\}}|S_t = s\} \\ &= \sum_{\omega_{t+1}\in\Omega_{t+1}} P(W_{t+1} = \omega_{t+1}) 1_{\{s'=S^M(S_t,x,\omega_{t+1})\}}. \end{aligned} \tag{9.17}$$

It is common in the field of Markov decision processes to assume that the one-step transition is given as data. Often, it can be quickly derived (for simple problems) using assumptions about the underlying process. For example, consider a financial asset selling problem with state variable $S_t = (R_t, p_t)$ where

$$R_t = \begin{cases} 1 & \text{we are still holding the asset,} \\ 0 & \text{the asset has been sold.} \end{cases}$$

and where p_t is the price at time t. We assume the price process is described by

$$p_t = p_{t-1} + \epsilon_t,$$

where ϵ_t is a random variable with distribution

$$\epsilon_t = \begin{cases} +1 & \text{with probability 0.3,} \\ 0 & \text{with probability 0.6,} \\ -1 & \text{with probability 0.1.} \end{cases}$$

Assume the prices are integer and range from 1 to 100. We can number our states from 0 to 100 using

$$\mathcal{S} = \{(0,-),(1,1),(1,2),\ldots,(1,100)\}.$$

We propose that our rule for determining when to sell the asset is of the form

$$X^{\pi}(R_t, p_t) = \begin{cases} \text{sell asset} & \text{if } p_t < \bar{p}, \\ \text{hold asset} & \text{if } p_t \geq \bar{p}. \end{cases}$$

Assume that $\bar{p} = 60$. A portion of the one-step transition matrix for the rows and columns corresponding to the state $(0, -)$ and $(1, 58), (1, 59), (1, 60), (1, 61), (1, 62)$ looks like

$$P^{60} = \begin{array}{c} \text{(0,-)} \\ \text{(1,58)} \\ \text{(1,59)} \\ \text{(1,60)} \\ \text{(1,61)} \\ \text{(1,62)} \end{array} \left[\begin{array}{cccccc} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & .1 & .6 & .3 & 0 \\ 0 & 0 & 0 & .1 & .6 & .3 \\ 0 & 0 & 0 & 0 & .1 & .9 \end{array} \right].$$

As we saw in chapter 14, this matrix plays a major role in the theory of Markov decision processes, although its value is more limited in practical applications. By representing the system dynamics as a one-step transition matrix, it is possible to exploit the rich theory surrounding matrices in general and Markov chains in particular.

In engineering problems, it is far more natural to develop the transition function first. Given this, it may be possible to compute the one-step transition matrix exactly or estimate it using simulation. The techniques in this book do not, in general, use the one-step transition matrix, but instead use the transition function directly. However, formulations based on the transition matrix provide a powerful foundation for proving convergence of both exact and approximate algorithms.

9.6.2 Model-free dynamic programming

There are many complex operational problems where we simply do not have a transition function. Some examples include

■ **EXAMPLE 9.1**

We are trying to find an effective policy to tax carbon to reduce CO2 emissions. We may try increasing the carbon tax, but the dynamics of climate change are so complex that the best we can do is wait a year and then repeat our measurements.

■ **EXAMPLE 9.2**

Uber encourages drivers to go on duty by raising prices (surge pricing). Since it is impossible to predict how drivers will behave, it is necessary to simply raise the price and observe how many drivers come on duty (or go off duty).

■ **EXAMPLE 9.3**

A utility managing a water reservoir can observe the level of the reservoir and control the release of water, but the level is also affected by rainfall, river inflows, and exchanges with ground water, which are unobservable.

These examples illustrate a problem where we do not know the dynamics, where the system reflects the unknown utility function of Uber drivers, and unobservable exogenous information. As a result, we either do not know the transition function itself, or there are decisions that we cannot model, or exogenous information we cannot simulation. In all three cases, we cannot compute the transition $S_{t+1} = S^M(S_t, x_t, W_{t+1})$.

In such settings (which are surprisingly common), we assume that given the state S_t, we make an action x_t and then simply observe the next state S_{t+1}. We can put this in the format of our original model by letting W_{t+1} be the new state, and writing our transition function as

$$S_{t+1} = W_{t+1}.$$

However, it is more natural (and compact) to simply assume that our system evolves according to

$$S_0 \to x_0 \to S_1 \to x_1 \to S_2 \to \ldots.$$

9.6.3 The resource transition function

There are many stochastic optimization problems that can be modeled in terms of managing "resources" where the state vector is denoted R_t. We use this notation when we want to specifically exclude other dimensions that might be in a state variable (for example, the challenge of making decisions to better estimate a quantity, which was first introduced in section 8.3). If we are using R_t as the state variable, the general representation of the transition function would be written

$$R_{t+1} = R^M(R_t, x_t, W_{t+1}).$$

Our notation is exactly analogous to the notation for a general state variable S_t, but it is sometimes useful to separate the modeling of resources (which are almost always controlled by our system) from other state variables that may evolve completely exogenously.

A simple example of a resource transition function arises in inventory planning, where R_t is the amount of resource at time t (water in a reservoir, product on a store shelf, blood in inventory), x_t is a decision to add to ($x_t > 0$) or subtract from ($x_t < 0$) the inventory, and $\hat{R}_{t+1}$ is exogenous changes (rainfall into the reservoir, theft from our product in inventory). The resource transition function would then be written

$$R_{t+1} = R_t + x_t + \hat{R}_{t+1}.$$

9.6.4 Exogenous transitions

There are many problems where some of the state variables evolve exogenously over time: rainfall, a stock price (assuming we cannot influence the price), the travel time on a congested road network, and equipment failures. There are two ways of modeling these processes.

The first models the change in the variable. If our state variable is a price p_t, we might let $\hat{p}_{t+1}$ be the change in the price between t and $t+1$, giving us the transition function

$$p_{t+1} = p_t + \hat{p}_{t+1}.$$

This has the advantage of giving us a clean transition function that describes how the price evolves over time. With this notation, we would write $W_{t+1} = (\hat{p}_{t+1})$, so that the exogenous information is distinct from the state variable.

Alternatively, we could simply assume that the new state p_{t+1} is the exogenous information. This means that we would write $W_{t+1} = p_{t+1}$ which means that we do not have a proper transition function to model the evolution of this particular state variable.

9.7 THE OBJECTIVE FUNCTION

The final dimension of our model is the objective function. We divide our discussion between creating performance metrics for evaluating a decision x_t, and evaluating the policy $X^\pi(S_t)$.

9.7.1 The performance metric

Performance metrics are described using a variety of terms such as

- Rewards, profits, revenues, costs (business)
- Gains, losses (engineering)
- Strength, conductivity, diffusivity (materials science)
- Tolerance, toxicity, effectiveness (health)
- Stability, reliability (engineering)
- Risk, volatility (finance)
- Utility (economics)
- Errors (machine learning)
- Time (to complete a task)

These differ primarily in terms of units and whether we are minimizing or maximizing. These are modeled using a variety of notation systems such as c for cost, r for revenue or reward, g for gain, L or ℓ for loss, U for utility, and $\rho(X)$ as a risk measure for a random variable X. We use two notations: $F(\cdot)$ as perhaps the most widely used notation for a function to be minimized or maximized, and $C(\cdot)$ for cost or contribution.

More interesting is what these functions depend on. Depending on the setting, we might use the following metrics:

$F(x, W)$ = A general performance metric (to be minimized or maximized) that depends only on the decision x and information W that is revealed after we choose x.

$C(S_t, x_t)$ = A cost/contribution function that depends on the state S_t and decision x_t.

$C(S_t, x_t, W_{t+1})$ = A cost/contribution function that depends on the state S_t and the decision x_t, and the information W_{t+1} that is revealed after x_t is determined.

$C(S_t, x_t, S_{t+1})$ = A cost/contribution function that depends on the state S_t and the decision x_t, after which we observe the subsequent state S_{t+1}.

We have used the notation $F(x, W)$ (as we did in chapters 5 and 7) when our problem does not depend on the state. However, as we transition to state-dependent problems, we will use $C(S_t, x_t)$ (or $C(S_t, x_t, W_{t+1})$ or $C(S_t, x_t, S_{t+1})$) to communicate that the objective function (or constraints or expectation) depend on the state. Readers may choose to use any notation such as $r(\cdot)$ for reward, $g(\cdot)$ for gain, $L(\cdot)$ for loss, or $U(\cdot)$ for utility.

The state-dependent representations all depend on the state S_t (or S^n if we wish), but it is useful to say what this means. When we make a decision, we need to work with a cost function and possibly constraints where we express the dependence on S_t by writing $\mathcal{X}_t$. For example, we might move money in a mutual fund to or from cash, buying or selling an index that is at price p_t. Let R_t be the amount of available cash, which evolves as people make deposits or withdrawals. The amount of cash could be defined by

$$\begin{aligned} R_{t+1}^{cash} &= R_t^{cash} + x_t + \hat{R}_{t+1}, && (9.18)\\ R_{t+1}^{index} &= R_t^{index} - x_t. && (9.19) \end{aligned}$$

where $x_t > 0$ is the amount of money moved into cash by selling the index fund, while $x_t < 0$ represents money from from cash into the index fund. We have to observe the constraints

$$\begin{aligned} x_t &\leq R_t^{index},\\ -x_t &\leq R_t^{cash}. \end{aligned}$$

The money we make is based on what we receive from buying or selling the index fund, which we would write as

$$C(S_t, x_t) = p_t x_t,$$

where the price evolves according to the model

$$p_{t+1} = \theta_0 p_t + \theta_1 p_{t-1} + \varepsilon_{t+1}.$$

For this problem, our state variable would be $S_t = (R_t, p_t, p_{t-1})$. In this problem, the contribution function itself depends on the state through the prices, while the constraints (R_t^{index} and R_t^{cash}) also vary dynamically and are part of the state.

Now imagine that we have to make the decision to buy or sell shares of our index fund, but the price we get is based on the closing price, which is not known when we make our decision. In this case, we would write our contribution function as

$$C(S_t, x_t, W_{t+1}) = p_{t+1} x_t,$$

where $W_{t+1} = \hat{p}_{t+1} = p_{t+1} - p_t$. We note that our policy $X^\pi(S_t)$ for making the decision x_t is not allowed to use W_{t+1}; rather, we have to wait until time $t+1$ before evaluating the quality of the decision.

Finally, consider a model of a hydroelectric reservoir where we have to manage the inventory in the reservoir, but where the dynamics describing its evolution is much more complicated than equations such as (9.18) and (9.19). In this setting, we can observe the reservoir level R_t, then make a decision of how much water to release out of the reservoir x_t, after which we observe the updated reservoir level R_{t+1}. This is similar to observing an updated price p_{t+1}. For these problems, we might let W_{t+1} be the new state, in which case our "transition equations" are just

$$S_{t+1} = W_{t+1}.$$

Alternatively, we may find it more natural to write the contribution function $C(S_t, x_t, S_{t+1})$, which is fairly common.

It should be apparent by now that we can model *all* the problems using the notation $C(S_t, x_t, W_{t+1})$, since this covers functions such as $F(x, W)$ that are not a function of the state, or $C(S_t, x_t)$ which are not a function of W_{t+1}. Finally, if we let $W_{t+1} = S_{t+1}$, then we also include $C(S_t, x_t, S_{t+1})$.

9.7.2 Finding the best policy

We close our first pass through modeling by giving the objective function for finding the best policy. In chapter 7, we made the distinction between terminal reward and cumulative reward formulations. For example, we might be interested in the value of an implementation decision $x^{\pi,N}$, which we can write

$$F^\pi(S_0) = \mathbb{E}_{S_0}\mathbb{E}_{W_1,\ldots,W_T|S_0}\mathbb{E}_{\widehat{W}|S_0}\{F(x^{\pi,T}, \widehat{W})|S_0\},$$

where $W^1, \ldots, W^N$ represent the observations we make while learning our decision (or design) $x^{\pi,T}$, while $\widehat{W}$ is the random variable we use for testing purposes.

If we wish to maximize cumulative rewards (where we have to learn by doing), we would write

$$F^\pi(S_0) \quad = \quad \mathbb{E}_{S_0}\mathbb{E}_{W_1,\ldots,W_T|S_0}\left\{\sum_{t=0}^{T} F(X^\pi(S_t), W_{t+1})|S_0\right\}.$$

For state-dependent problems, we can write the cumulative reward formulation by replacing $F(X^\pi(S_t), W_{t+1})$ with $C(S_t, X^\pi(S_t), W_{t+1})$.

We can write our terminal reward formulation if we use a time-dependent contribution function $C_t(S_t, X^\pi(S_t), W_{t+1})$ defined by

$$C_t(S_t, X^\pi(S_t), W_{t+1}) = \left\{ \begin{array}{cc} 0 & t < T, \\ F(x^{\pi,T}, \widehat{W}) & t = T. \end{array}\right.$$

where $\widehat{W}$ is the random variable we use for testing a design for offline (terminal reward) settings. This allows us to write

$$\begin{aligned} F^\pi(S_0) &= \mathbb{E}_{S_0}\mathbb{E}_{W_1,\ldots,W_T,W_{T+1}|S_0}\left\{\sum_{t=0}^{T} C_t(S_t, X^\pi(S_t), W_{t+1})|S_0\right\} \\ &= \mathbb{E}_{S_0}\mathbb{E}_{W_1,\ldots,W_T,W_{T+1}|S_0} C_T(S_T, X^\pi(S_T), W_{T+1}) \end{aligned}$$

where we assume that $W_{T+1} = \widehat{W}$. If we are using the cumulative reward objective, W_{T+1} is just ignored since $C_T(S_T, X^\pi(S_T), W_{T+1}) = C_T(S_T, X^\pi(S_T))$ in this setting.

Thus, we see that if we write our contribution function as $C_t(S_t, x_t, W_{t+1})$ (or $C^n(S^n, x^n, W^{n+1})$ if we are indexing by iteration n), then we can model any of the situations listed above. For this reason, we propose as our *universal objective function* the following:

$$\max_{\pi\in\Pi} \mathbb{E}_{S_0}\mathbb{E}_{W_1,\ldots,W_{T+1}|S_0}\left\{\sum_{t=0}^{T} C_t(S_t, X_t^\pi(S_t), W_{t+1})|S_0\right\}. \tag{9.20}$$

We refer to equation (9.20) as our universal objective function since we can model state-dependent and state-independent problems, covering both final reward and cumulative reward. We revisit these different problem classes in section 9.10 below. Next, we illustrate our modeling framework using an energy storage problem.

9.8 ILLUSTRATION: AN ENERGY STORAGE MODEL

We are going to use a simple energy storage application to illustrate the process of modeling a dynamic program, following the canonical framework described in this chapter, consisting of states, actions, exogenous information, transition function and objective function. While we list the five elements in this order, we are going to describe the state variable last, since it consists of all the information needed to compute the cost function, decision function, and transition function. For this purpose, it is more natural to describe these functions first, and then pull the necessary information together into the state variable.

Decision/control variable:
In our storage system, let G refer to grid, E refer to our renewable energy, B is the "battery" (storage), and L be the load (the demand for power). We would then designate, for example, the flow of energy from the grid to the load as x_t^{GL}. Our control is the vector

$$x_t \quad = \quad (x_t^{GL}, x_t^{GB}, x_t^{EL}, x_t^{EB}, x_t^{BL}).$$

The decisions are subject to several constraints:

$$\begin{aligned} x_t^{EL} + x_t^{EB} &\leq E_t, & (9.21)\\ x_t^{GL} + x_t^{EL} + \eta_t x_t^{BL} &= L_t, & (9.22)\\ x_t^{BL} &\leq R_t, & (9.23)\\ x^{EB}, x^{GB}, x^{BL} &\leq \rho^{chrg}, & (9.24)\\ x_t^{GL}, x_t^{EL}, x_t^{EB}, x_t^{BL} &\geq 0. & (9.25) \end{aligned}$$

Equation (9.21) limits the energy from renewables to what is being generated, while (9.22) limits the amount that we can send to the customer by the load at that point in time. Equation (9.23) limits what we can draw from the battery to what is in the battery at time t. Equation (9.24) limits the charge/discharge rate for the battery (we assume that ρ^{chrg} is a fixed parameter). Note that we apply nonnegativity to every flow variable except x_t^{GB}, since we can sell to the grid. We refer to the feasible region defined by (9.21)-(9.25) as $\mathcal{X}_t$, where this is implicitly a function of S_t.

If we use a lookahead policy, we will need any available forecasts. We represent our forecast of loads using

$$\begin{aligned} f_{tt'}^L &= \text{Forecast of the load } L_{t'} \text{ at time } t' > t \text{ given what we know at time } t,\\ &= \mathbb{E}_t L_{t'},\\ f_t^L &= (f_{tt'}^L)_{t'>t}. \end{aligned}$$

Similarly, we might have forecasts of energy from renewables f_t^E, forecasts of exogenous inputs f_t^R (represented as $\hat{R}_t$ below), and forecasts of losses f_t^η. We can represent our set of forecasts using

$$f_t = (f_t^L, f_t^E, f_t^R, f_t^\eta).$$

Exogenous information:
Our problem evolves in the presence of the following exogenous information processes:

$$\begin{aligned}
\hat{R}_t &= \text{Exogenous change to the energy stored between } t-1 \text{ and } t \text{ (e.g., rainfall, chemical leakage),}\\
\epsilon_t^p &= \text{A noise term in the transition equation for prices (see below) that is revealed at time } t,\\
\hat{E}_t &= \text{The change in energy produced from renewable sources between } t-1 \text{ and } t,\\
\hat{L}_t &= \text{Deviation from the forecasted load between } t-1 \text{ and } t,\\
\hat{\eta}_t &= \text{Change in the rate of energy conversion loss (e.g., due to temperature) between } t-1 \text{ and } t.
\end{aligned}$$

We assume that our vector of forecasts f_t is provided by an exogenous (commercial) forecasting service, so we are given the forecast directly (rather than the change in the forecast). Our exogenous information W_t is then

$$W_t \quad = \quad (\hat{R}_t, \epsilon_t^p, \hat{E}_t, \hat{L}_t, \hat{\eta}_t, f_t).$$

To complete the model, we would have to either provide the probability distribution for the exogenous variables, or to specify the source for actual observations.

Transition function:
The equations describing the evolution of the state variables are given by

$$R_{t+1} = R_t + \eta_{t+1}(x_t^{GB} + x_t^{EB}) - x_t^{BL} + \hat{R}_{t+1}, \tag{9.26}$$

$$p_{t+1} = \theta_0 p_t + \theta_1 p_{t-1} + \theta_2 p_{t-2} + \epsilon_{t+1}^p, \tag{9.27}$$

$$E_{t+1} = E_t + \hat{E}_{t+1}, \tag{9.28}$$

$$L_{t+1} = L_t + \hat{L}_{t+1}, \tag{9.29}$$

$$\eta_{t+1} = \eta_t + \hat{\eta}_{t+1}. \tag{9.30}$$

In equation (9.27), ϵ_{t+1}^p is a random variable independent of any history. These equations represent the transition function $S^M(S_t, x_t, W_{t+1})$. Note that our resource transition function depends on stochastic losses η_{t+1} (such as evaporation) and exogenous input $\hat{R}_{t+1}$ (such as rainfall) that only become known at time $t+1$.

Objective function:
Our cost function captures what we pay (or receive) when we buy from or discharge to the grid at the current price p_t

$$C(S_t, x_t) = p_t x_t^{GB},$$

where we pay money if we are charging our device ($x_t > 0$) and we receive money if we are selling back to the grid ($x_t < 0$).

The objective function is the canonical objective originally given in equation (14.1):

$$\min_{\pi \in \Pi} \mathbb{E}^\pi \sum_{t=0}^{T} C(S_t, X_t^\pi(S_t)), \tag{9.31}$$

where $S_{t+1} = S^M(S_t, X_t^\pi(S_t), W_{t+1})$, which is given by equations (9.26)-(9.30).

We are now ready to define our state variable.

State variable: We are now ready to pull all the information we need, which defines the state variable. The decision function needs the following information in the constraints:

R_t = The amount of energy in storage at time t,

E_t = The level of energy from renewables at time t,

L_t = The load at time t,

η_t = The rate of energy loss (e.g., due to evaporation or temperature) between $t-1$ and t.

The transition function depends on the current price p_t, as well as the two recent prices p_{t-1} and p_{t-2}.

We would also point out that the transition function generally needs access to forecasts. For example, we might sample the load L_{t+1} using

$$L_{t+1} = f^L_{t,t+1} + \varepsilon^L_{t+1}.$$

In addition, we have to update the entire set of foreward forecasts using

$$f_{t+1,t'} = f_{tt'} + \varepsilon_{t'}.$$

Thus, forecasts are needed for the transition function, in addition to possibly being needed in a lookahead policy.

The cost function needs only p_t:

p_t = The price of electricity from the grid at time t.

We can now pull all of the necessary information together to form our state variable

$$S_t = \big(R_t, (p_t, p_{t-1}, p_{t-2}), E_t, L_t, \eta_t, f_t\big).$$

9.9 BASE MODELS AND LOOKAHEAD MODELS

There is a subtle but critical distinction between a "model" of a real problem, and what we will come to know as a "lookahead model," which is an approximation which is used to peek into the future (typically with various convenient approximations) for the purpose of making a decision now. We are going to describe lookahead models in far greater depth in chapter 20, but we feel that it is useful to make the distinction now.

Using the framework presented in this chapter, we can write almost any sequential decision process in the compact form

$$max_{\pi\in\Pi}\mathbb{E}^\pi\left\{\sum_{t=0}^{T} C_t(S_t, X^\pi_t(S_t), W_{t+1})|S_0\right\}$$

where $S_{t+1} = S^M(S_t, X^\pi_t(S_t), W_{t+1})$. Of course, we have to specify our model for $(W_t)_{t=0}^{T+1}$ in addition to defining the state variable (later we will address the issue of identify our class of policies).

For the moment, we view (9.32) (along with the transition function) as "the problem" that we are trying to solve. If we find an effective policy, we assume we have solved "the

	Offline Terminal reward	Online Cumulative reward
State independent problems	$\max_\pi \mathbb{E}\{F(x^{\pi,N},W)\|S_0\}$ Stochastic search (1)	$\max_\pi \mathbb{E}\{\sum_{n=0}^{N-1} F(X^\pi(S^n),W^{n+1})\|S_0\}$ Multiarmed bandit problem (2)
State dependent problems	$\max_{\pi^{lrn}} \mathbb{E}\{C(S,X^{\pi^{imp}}(S\|\theta^{imp}),W)\|S_0\}$ Offline dynamic programming (4)	$\max_\pi \mathbb{E}\{\sum_{t=0}^{T} C(S_t,X^\pi(S_t),W_{t+1})\|S_0\}$ Online dynamic programming (3)

Table 9.2 Comparison of formulations for state-independent (learning) vs. state-dependent problems, and offline (terminal reward) and online (cumulative reward).

problem." However, we are going to learn that in dynamic systems, we are often solving a problem at some time t over a horizon $(t,\ldots,t+H)$, where we simply set $t=0$ and number time periods accordingly. The question is: are we interested in the solution over the entire planning horizon, or just the decision in the first time period?

Given the widespread use of lookahead models, we need a term to identify when we are presenting a model of a problem we wish to solve. We might use the term "real model" to communicate that this is our model of the real world. Statisticians use the term "true model," but this seems to assume that we have somehow perfectly modeled a real problem, which is never the case. Some authors use the term "nominal model," but we feel that this is not sufficiently descriptive.

In this book, we use the term *base model* since we feel that this communicates the idea that this is the model we wish to solve. We take the position that regardless of any modeling approximations that have been introduced (either for reasons of tractability or availability of data), this is "the" model we are trying to solve.

Later, we are going to introduce approximations of our base model, which may still be quite difficult to solve. Most important will be the use of lookahead models, which we discuss in depth in chapter 20.

9.10 A CLASSIFICATION OF PROBLEMS*

It is useful to contrast problems based on two key dimensions: First, whether the objective function is offline (terminal reward) or online (cumulative reward), and second, whether the objective function is state-independent (learning problems, which we covered in chapters 5 and 7) or state-dependent (traditional dynamic programs), which we began treating in chapter 8, and which will be the focus of the remaining chapters.

This produces four problem classes which are depicted in table 9.2. Moving clockwise around the table, starting from the upper left-hand corner:

Class 1) State-independent, terminal reward - This describes classical search problems where we are trying to find the best algorithm (which we call a policy π) for finding the best solution $x^{\pi,N}$ within our budget N. After n experiments the state S^n captures only our belief state about the function $\mathbb{E}F(x,W)$, and our decisions are made with a policy (or algorithm) $x^n = X^\pi(S^n)$. We can write this problem as

$$\max_\pi \mathbb{E}\{F(x^{\pi,N},\widehat{W})|S^0\} = \mathbb{E}_{S^0}\mathbb{E}_{W^1,\ldots,W^N|S^0}\mathbb{E}_{\widehat{W}|S^0}F(x^{\pi,N},\widehat{W}), \tag{9.32}$$

where $W^1, \ldots, W^N$ are the observations of W while learning the function $\mathbb{E}F(x, W)$, and $\widehat{W}$ is the random variable used for testing the final design $x^{\pi,N}$. The distinguishing characteristics of this problem are a) that the function $F(x, W)$ depends only on x and W, and not on the state S^n, and b) that we evaluate our policy $X^\pi(S)$ only after we have exhausted our budget of N experiments. We do allow the function $F(x, W)$, the observations $W^1, \ldots, W^N$ and the random variable $\widehat{W}$ to depend on the initial state S_0, which includes any deterministic parameters, as well as probabilistic information (such as a Bayesian prior) that describes any unknown parameters (such as how the market responds to price).

Class 2) State-independent, cumulative reward - Here we are looking for the best policy that learns while it optimizes. This means that we are trying to maximize the sum of the rewards received within our budget. This is the classic multiarmed bandit problem that we first saw in chapter 7 if the decisions x were discrete and we did not have access to derivatives (but we are not insisting on these limitations). We can write the problem as

$$\max_\pi \mathbb{E} \left\{ \sum_{n=0}^{N-1} F(X^\pi(S^n), W^{n+1}) | S^0 \right\} = \mathbb{E}_{S^0} \mathbb{E}_{W^1, \ldots, W^N | S^0} \sum_{n=0}^{N-1} F(X^\pi(S^n), W^{n+1}). \tag{9.33}$$

Class 3) State-dependent, cumulative reward - We now transition to problems where we are maximizing contributions that depend on the state variable, the decision, and possibly (but not always) random information that arrives after we make a decision (if it arrived before, it would be included in the state variable). For this reason, we are going to switch from our notation $F(x, W)$ to our notation $C(S, x, W)$ (or, in a time-indexed environment, $C(S_t, x_t, W_{t+1})$). As with the multiarmed bandit problem (or more generally, Class (2) problems), we want to find a policy that learns while implementing. These problems can be written

$$\max_\pi \mathbb{E} \left\{ \sum_{t=0}^{T} C(S_t, X^\pi(S_t), W_{t+1}) | S_0 \right\} = \mathbb{E}_{S_0} \mathbb{E}_{W_1, \ldots, W_T | S_0} \left\{ \sum_{t=0}^{T} C(S_t, X^\pi(S_t), W_{t+1}) | S_0 \right\}. \tag{9.34}$$

State variables in this problem class may include any of the following:

- Variables that are controlled (or influenced) by decisions (such as inventory or the location of a sensor on a graph). These variables directly affect the contribution function (such as price) or the constraints (such as the inventory).
- Variables that evolve exogenously (such as the wind speed or price of an asset).
- Variables that capture our belief about a parameter that are only used by the policy.

When we consider that our state S_t may include a controllable physical state R_t, exogenous information I_t and/or a belief state B_t, we see that this covers a very broad range of problems. The key feature here is that our policy has to maximize cumulative contributions as we progress, which may include learning (if there is a belief state).

Class 4) State-dependent, terminal reward - For our state-independent function $F(x, W)$ we were looking for the best policy to learn the decision $x^{\pi,N}$ to be implemented. In this setting, we can think of the policy as a *learning policy*, while $x^{\pi,N}$ is the *implementation decision*. In the state-dependent case, the implementation decision becomes one that depends on the state (at least, part of the state), which is a function we call the *implementation policy*. We designate the implementation policy by $X^{\pi^{imp}}(S|\theta^{imp})$, which we write as depending on a set of parameters θ^{imp} which have to be learned. We designate the learning policy for learning θ^{imp} by $\Theta^{\pi^{lrn}}(S|\theta^{lrn})$ which proceeds by giving us parameters $\theta^{imp,n} = \Theta^{\pi^{lrn}}(S^n|\theta^{lrn})$. The problem can be written

$$\begin{aligned}\max_{\pi^{lrn}} \mathbb{E}\{C(S, X^{\pi^{impl}}(S|\theta^{imp}), \widehat{W})|S^0\} \quad &= \\ \mathbb{E}_{S^0}\mathbb{E}^{\pi^{lrn}}_{W^1,\ldots,W^N|S^0}\mathbb{E}^{\pi^{imp}}_{S|S^0}\mathbb{E}_{\widehat{W}|S^0}C(S, X^{\pi^{impl}}&(S|\theta^{imp}), \widehat{W}). \qquad (9.35)\end{aligned}$$

where $W^1, \ldots, W^N$ represents the observations made while using our budget of N experiments to learn a policy, and $\widehat{W}$ is the random variable observed when evaluating the policy at the end. We use the expectation operator $\mathbb{E}^{\pi^{lrn}}$ indexed by the learning policy when the expectation is over a random variable whose distribution is affected by the learning policy.

The learning policy could be a stochastic gradient algorithm to learn the parameters θ^{imp}, or it could be one of our derivative-free methods such as interval estimation or upper confidence bounding. The learning policy could be algorithms for learning value functions such as Q-learning (see equations (2.16)-(2.17) in chapter 2), or the parameters of any of the derivative-free search algorithms in chapter 7.

We typically cannot compute the expectation $\mathbb{E}^{\pi^{imp}}_S$ since it depends on the implementation policy which in turn depends on the learning policy. As an alternative, we can run a simulation over a horizon $t = 0, \ldots, T$ and then divide by T to get an average contribution per unit time. Let $W^n = (W^n_1, \ldots, W^n_T)$ be a simulation over our horizon. This allows us to write our learning problem as

$$\max_{\pi^{lrn}} \mathbb{E}_{S^0}\mathbb{E}^{\pi^{imp}}_{((W^n_t)^T_{t=0})^N_{n=0}|S^0}\left(\mathbb{E}^{\pi^{imp}}_{(\widehat{W}_t)^T_{t=0}|S^0}\frac{1}{T}\sum_{t=0}^{T-1} C(S_t, X^{\pi^{imp}}(S_t|\theta^{imp}), \widehat{W}_{t+1})\right). \qquad (9.36)$$

This parallels class (1) problems. We are searching over learning policies which determines the implementation policy through $\theta^{imp} = \Theta^{\pi^{lrn}}(S|\theta^{lrn})$, where the simulation over time replaces $F(x, W)$ in the state-independent formulation. The sequence $(W^n_t)^T_{t=0}, n = 1, \ldots, N$ replaces the sequence $W^1, \ldots, W^N$ for the state-independent case, where we start at state $S_0 = S^0$. We then do our final evaluation by taking an expectation over $(\widehat{W}_t)^T_{t=0}$, where we again assume we start our simulations at $S^0 = S_0$.

9.11 POLICY EVALUATION*

While it is certainly useful to characterize these four problem classes, it is an entirely different matter to compute the expectations in equations (9.32)-(9.36). The best way to approach this task (in fact, the best way to actually understand the expectations) is to simulate them. In this section we describe how to approximate each expectation using simulation.

We begin by fixing a policy $X^\pi(S_t|\theta)$ parameterized by some vector θ, which can be anything including a learning policy such as Thompson sampling, a stochastic gradient algorithm with a particular stepsize policy, or a direct lookahead policy. In problem class (1), $X^\pi(S_t|\theta)$ is a pure learning policy which learns an implementation decision $x^{\pi,N}(\theta)$. In classes (2) and (3), it is a policy where we learn as we implement. In class (4), we use a learning policy $\Theta^{\pi^{lrn}}(S_t|\theta^{lrn})$ to learn the parameter θ^{imp} of an implementation policy $X^{\pi^{imp}}(S_t|\theta^{imp})$ where $\theta^{imp} = \Theta^{\pi^{lrn}}(\theta^{lrn})$ depends on the learning policy.

Throughout, we are going to use θ (or θ^{lrn}) as a (possibly vector-valued) parameter that controls our learning policy (for classes (1) and (4)) or the implementation policy (possibly with learning) for classes (2) and (3). The vector θ (or θ^{lrn}) might be the parameters governing the behavior of any adaptive learning algorithm.

We now need to evaluate how well this policy works. We start with state S^0 if we are in problem classes (1) or (4), or S_0 if we are in problem class (3), and S^0 or S_0 if we are in problem class (2). From the initial state, we pick initial values of any parameters, either because they are fixed, or by drawing them from an assumed distribution (that is, a Bayesian prior).

We next address the process of simulating a policy for each of the four problem class.

Class 1) State-independent, final reward - From an initial state S^0, we use our (learning) policy to make decision $x^0 = X^\pi(S^0|\theta)$, and then observe outcome W^1, producing an updated state S^1 (in this problem, S^n is a pure knowledge state). The parameter θ controls the behavior of our learning policy. We repeat this until our budget is depleted, during which we observe the sequence $W^1(\omega), \ldots, W^N(\omega)$, where we let ω represent a particular sample path. At the end we learn state S^N, from which we find our best solution (the final design) $x^{\pi,N}$, which we write as $x^{\pi,N}(\theta|\omega)$ to express its dependence on the learning policy π (parameterized by θ) and the sample path ω.

We then evaluate $x^{\pi,N}(\theta|\omega)$ by simulating $\mathbb{E}_{\widehat{W}} F(x^{\pi,N}(\theta|\omega), \widehat{W})$ by repeatedly sampling from $\widehat{W}$ to get sampled estimates of $\mathbb{E}_{\widehat{W}} F(x^{\pi,N}(\theta|\omega), \widehat{W})$. Let $\widehat{W}(\psi)$ be a particular realization of $\widehat{W}$. A sampled estimate of the policy π (which we assume is parameterized by θ) is given by

$$F^\pi(\theta|\omega, \psi) = F(x^{\pi,N}(\theta|\omega), \widehat{W}(\psi)). \tag{9.37}$$

We now average over a set of K samples of ω, and L samples of ψ, giving us

$$\overline{F}^\pi(\theta) = \frac{1}{K}\frac{1}{L}\sum_{k=1}^{K}\sum_{\ell=1}^{L} F^\pi(\theta|\omega^k, \psi^\ell). \tag{9.38}$$

Class 2) State-independent, cumulative reward - This problem can be interpreted in two ways. As the cumulative reward version of problem class (1), we simulate our policy for N iterations, giving us the sequence $(S^0, x^0, W^1, \ldots, x^{N-1}, W^N, S^N)$. Here, we accumulate our rewards, producing a sampled estimate

$$F^\pi(\theta|\omega) = \sum_{n=0}^{N-1} F(X^\pi(S^n|\theta), W^{n+1}(\omega)). \tag{9.39}$$

Unlike class (1), we evaluate our policy as we go, avoiding the need for the final step at the end. We would then compute an average using

$$\overline{F}^{\pi}(\theta) = \frac{1}{K}\sum_{k=1}^{K} F^{\pi}(\theta|\omega^k), \tag{9.40}$$

over a sample of K observations.

We can also recast this problem as simulating over time, where we just replace W^n with W_t and S^n with S_t.

Class 3) State-dependent, cumulative reward - This is the state-dependent version of problem class (2), which we model as evolving over time. Starting in state S_0, we simulate the policy much as we did in equation (9.39) which is given by

$$F^{\pi}(\theta|\omega) = \sum_{t=0}^{T-1} C(S_t(\omega), X^{\pi}(S_t(\omega)|\theta), W_{t+1}(\omega)). \tag{9.41}$$

We then average over sample paths to obtain

$$\overline{F}^{\pi}(\theta) = \frac{1}{K}\sum_{k=1}^{K} F^{\pi}(\theta|\omega^k). \tag{9.42}$$

Class 4) State-dependent, final reward - We now have a hybrid of problem classes (1) and (3), where we use a learning policy $\Theta^{\pi^{lrn}}(S|\theta^{lrn})$ to learn the parameters of an implementation policy $X^{\pi^{imp}}(S_t|\theta^{imp})$, where the parameter $\theta^{imp} = \Theta^{\pi^{lrn}}(\theta^{lrn})$ that determines the behavior of the implementation policy depends on the learning policy π^{lrn} and its tunable parameters θ^{lrn}. We then have to evaluate the implementation policy, just as we evaluated the final design $x^{\pi,N}(\theta)$ in class (1), where $x^{\pi,N}(\theta)$ is the implementation decision that depends on the learning policy π and its parameters θ.

In class (1), we evaluated the implementation decision $x^{\pi,N}(\theta)$ by simulating $\widehat{W}$ to obtain estimates of $F(x^{\pi,N}, \widehat{W})$. Now we have to take an expectation over the state S which we do by simulating our implementation policy $X^{\pi^{imp}}(S_t|\theta^{imp})$ starting in state S_0 until the end of our horizon S_T. One simulation from 0 to T is comparable to an evaluation of $F(x, W)$. This means that a sample path ω, which in (1) was one observation of $W_1, \ldots, W_T$, is an observation of $(W_t^n, t = 1, \ldots, T), n = 0, \ldots, N$. This observation then produces the implementation policy $X^{\pi^{imp}}(S_t|\theta^{imp})$ (whereas in class (1) problems it produced the implementation decision $x^{\pi,N}(\theta|\omega)$).

To simulate the value of the policy, we simulate one last set of observations $\widehat{W}_1(\psi), \ldots, \widehat{W}_T(\psi)$ which, combined with our implementation policy which we write as $X^{\pi^{imp},N}(S_t|\theta^{imp}, \omega)$ produces a sequence of states $S_t(\psi)$, giving us the estimate

$$F^{\pi}(\theta^{lrn}|\omega, \psi) = \sum_{t=0}^{T} C(S_t(\psi), X^{\pi^{imp}}(S_t(\psi)|\theta^{imp}, \omega), \widehat{W}_t(\psi)), \tag{9.43}$$

where we need to remember that $\theta^{imp} = \Theta^{\pi^{lrn}}(\theta^{lrn})$. We finally average over a set of K samples of ω, and L samples of ψ, giving us

$$\overline{F}^{\pi}(\theta^{lrn}) = \frac{1}{K}\frac{1}{L}\sum_{k=1}^{K}\sum_{\ell=1}^{L} F^{\pi}(\theta^{lrn}|\omega^k, \psi^\ell), \tag{9.44}$$

We now have a way of computing the performance of a policy $\overline{F}^{\pi}(\theta)$, which may be a learning policy for classes (1) and (4), or an implementation (and learning) policy for classes (2) and (3).

9.12 ADVANCED PROBABILISTIC MODELING CONCEPTS**

Dynamic programs introduce some very subtle issues when bridging with classical probability theory. This material is not important for readers who just want to focus on models and algorithms. However, understanding how the probability community thinks of stochastic dynamic programs provides a fresh perspective that brings a deep pool of theory from the probability community.

9.12.1 A measure-theoretic view of information

For researchers interested in proving theorems or reading theoretical research articles, it is useful to have a more fundamental understanding of information.

When we work with random information processes and uncertainty, it is standard in the probability community to define a probability space, which consists of three elements. The first is the set of outcomes Ω, which is generally assumed to represent all possible outcomes of the information process (actually, Ω can include outcomes that can never happen). If these outcomes are discrete, then all we would need is the probability of each outcome $p(\omega)$.

It is nice to have a terminology that allows for continuous quantities. We want to define the probabilities of our events, but if ω is continuous, we cannot talk about the probability of an outcome ω. However we can talk about a set of outcomes $\mathcal{E}$ that represent some specific event (if our information is a price, the event $\mathcal{E}$ could be all the prices that constitute the event that the price is greater than some number). In this case, we can define the probability of an outcome $\mathcal{E}$ by integrating the density function $p(\omega)$ over all ω in the event $\mathcal{E}$.

Probabilists handle continuous outcomes by defining a set of events $\mathfrak{F}$, which is literally a "set of sets" because each element in $\mathfrak{F}$ is itself a set of outcomes in Ω. This is the reason we resort to the script font $\mathfrak{F}$ as opposed to our calligraphic font for sets; it is easy to read $\mathcal{E}$ as "calligraphic E" and $\mathfrak{F}$ as "script F." The set $\mathfrak{F}$ has the property that if an event $\mathcal{E}$ is in $\mathfrak{F}$, then its complement $\Omega \setminus \mathcal{E}$ is in $\mathfrak{F}$, and the union of any two events $\mathcal{E}_X \cup \mathcal{E}_Y$ in $\mathfrak{F}$ is also in $\mathfrak{F}$. $\mathfrak{F}$ is called a "sigma-algebra" (which may be written "σ-algebra"), and is a countable union of events in Ω. An understanding of sigma-algebras is not important for computational work, but can be useful in certain types of proofs, as we see in the "why does it work" sections at the end of several chapters. Sigma-algebras are without question one of the more arcane devices used by the probability community, but once they are mastered, they are a powerful theoretical tool (but useless computationally).

Finally, it is required that we specify a probability measure denoted $\mathcal{P}$, which gives the probability (or density) of an outcome ω which can then be used to compute the probability of an event in $\mathfrak{F}$.

We can now define a formal probability space for our exogenous information process as $(\Omega, \mathfrak{F}, \mathcal{P})$, sometimes known as the "holy trinity" in probability. If we wish to take an expectation of some quantity that depends on the information, say $Ef(W)$, then we would sum (or integrate) over the set ω multiplied by the probability (or density) $\mathcal{P}$.

This notation is especially powerful for "static" problems where there are two points in time: before we see the random variable W, and after. This creates a challenge when we have sequential problems where information evolves over time. Probabilists have adapted the original concept of probability spaces $(\Omega, \mathfrak{F}, \mathcal{P})$ by manipulating the set of events $\mathfrak{F}$.

It is important to emphasize that ω represents *all* the information that will become available, over all time periods. As a rule, we are solving a problem at time t, which means we do not have the information that will become available after time t. To handle this, we let $\mathfrak{F}_t$ be the sigma-algebra representing events that can be created using only the information up to time t. To illustrate, consider an information process W_t consisting of a single 0 or 1 in each time period. W_t may be the information that a customer purchases a jet aircraft, or the event that an expensive component in an electrical network fails. If we look over three time periods, there are eight possible outcomes, as shown in table 9.3.

Outcome	Time period		
ω	1	2	3
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1

Table 9.3 Set of demand outcomes

Let $\mathcal{E}_{\{W_1\}}$ be the set of outcomes ω that satisfy some logical condition on W_1. If we are at time $t = 1$, we only see W_1. The event $W_1 = 0$ would be written

$$\mathcal{E}_{\{W_1=0\}} = \{\omega | W_1 = 0\} = \{1, 2, 3, 4\}.$$

The sigma-algebra $\mathfrak{F}_1$ would consist of the events

$$\{\mathcal{E}_{\{W_1=0\}}, \mathcal{E}_{\{W_1=1\}}, \mathcal{E}_{\{W_1 \in \{0,1\}\}}, \mathcal{E}_{\{W_1 \notin \{0,1\}\}}\}.$$

Now assume that we are at time $t = 2$ and have access to W_1 and W_2. With this information, we are able to divide our outcomes Ω into finer subsets. Our history H_2 consists of the elementary events $\mathcal{H}_2 = \{(0,0), (0,1), (1,0), (1,1)\}$. Let $h_2 = (0,1)$ be an element of H_2. The event $\mathcal{E}_{\{h_2=(0,1)\}} = \{3,4\}$. At time $t = 1$, we could not tell the difference between outcomes 1, 2, 3, and 4; now that we are at time 2, we can differentiate between $\omega \in \{1,2\}$ and $\omega \in \{3,4\}$. The sigma-algebra $\mathfrak{F}_2$ consists of all the events $\mathcal{E}_{h_2}, h_2 \in \mathcal{H}_2$, along with all possible unions and complements.

Another event in $\mathfrak{F}_2$ is $\{\omega|(W_1, W_2) = (0, 0)\} = \{1, 2\}$. A third event in $\mathfrak{F}_2$ is the union of these two events, which consists of $\omega = \{1, 2, 3, 4\}$ which, of course, is one of the events in $\mathfrak{F}_1$. In fact, every event in $\mathfrak{F}_1$ is an event in $\mathfrak{F}_2$, but not the other way around. The reason is that the additional information from the second time period allows us to divide Ω into finer set of subsets. Since $\mathfrak{F}_2$ consists of all unions (and complements), we can always take the union of events, which is the same as ignoring a piece of information. By contrast, we cannot divide $\mathfrak{F}_1$ into a finer subsets. The extra information in $\mathfrak{F}_2$ allows us to filter Ω into a finer set of subsets than was possible when we only had the information through the first time period. If we are in time period 3, $\mathfrak{F}$ will consist of each of the individual elements in Ω as well as all the unions needed to create the same events in $\mathfrak{F}_2$ and $\mathfrak{F}_1$.

From this example, we see that more information (that is, the ability to see more elements of $W_1, W_2, \ldots$) allows us to divide Ω into finer-grained subsets. For this reason, we can always write $\mathfrak{F}_{t-1} \subseteq \mathfrak{F}_t$. $\mathfrak{F}_t$ always consists of every event in $\mathfrak{F}_{t-1}$ in addition to other finer events. As a result of this property, $\mathfrak{F}_t$ is termed a *filtration*. It is because of this interpretation that the sigma-algebras are typically represented using the script letter F (which literally stands for filtration) rather the more natural letter H (which stands for history). The fancy font used to denote a sigma-algebra is used to designate that it is a set of sets (rather than just a set).

It is *always* assumed that information processes satisfy $\mathfrak{F}_{t-1} \subseteq \mathfrak{F}_t$. Interestingly, this is not always the case in practice. The property that information forms a filtration requires that we never "forget" anything. In real applications, this is not always true. Assume, for example, that we are doing forecasting using a moving average. This means that our forecast f_t might be written as $f_t = (1/T)\sum_{t'=1}^{T} \hat{D}_{t-t'}$. Such a forecasting process "forgets" information that is older than T time periods.

By far the most widespread use of the notation $\mathfrak{F}_t$ is to represent the information we know at time t. For example, let W_{t+1} be the information that we will learn at time $t + 1$. If we are sitting at time t, we might use a forecast $f^W_{t,t+1}$ which would be written

$$f^W_{t,t+1} = \mathbb{E}\{W_{t+1}|\mathfrak{F}_t\}. \tag{9.45}$$

Conditioning on $\mathfrak{F}_t$ means conditioning on what we know at time t which some authors will write

$$f^W_{t,t+1} = \mathbb{E}_t W_{t+1}. \tag{9.46}$$

Equations (9.45) and (9.46) are equivalent, and both would be read "the conditional expectation of W_{t+1} given what we know at time t."

If we do not include this conditioning, then this is the same as an expectation we would make at time 0, which we could write

$$\begin{aligned} f^W_{0,t+1} &= \mathbb{E}W_{t+1} \\ &= \mathbb{E}\{W_{t+1}|\mathfrak{F}_0\}. \end{aligned}$$

There are numerous textbooks on measure theory. For a nice introduction to measure-theoretic thinking (and in particular the value of measure-theoretic thinking), see Pollard (2002) for an introduction to measure-theoretic probability, or the advanced text Cinlar (2011). For an illustration of mathematics using this notation, see the "More modern proof" of convergence for stochastic gradient algorithms in section 5.8.3.

9.12.2 Conditional expectations for sequential decision problems**

In this section, we are going to address a fundamental problem with the mathematics of conditional expectation. If you do not have a problem with the expectation form of Bellman's equation (which we have replicated in equation (9.48) below), this section will not add any value to your understanding of dynamic programming. But if you are bothered by our habit of conditioning on the random variable S_t, read on. In particular, if you are amused by some of the subtle arguments that arise among theoretical probabilists, you may enjoy this section.

By now we have presented Bellman's optimality equation in several forms. In chapter 2, we wrote the form that is most widely seen in the literature where it is written

$$V_t(S_t) = \max_{x \in \mathcal{X}_s} \big(r(S_t, x) + \sum_{s' \in \mathcal{S}} P(s'|S_t, x) V_{t+1}(s') \big). \tag{9.47}$$

Here, we assume that the one-step transition matrix $P(s'|S_t, x)$ is known (in practice this is typically very hard to compute). In chapter 7, we write it in the equivalent expectation form (equation (7.19)) using

$$V_t(S_t) \quad = \quad \max_{x_t \in \mathcal{X}_t} \big(C_t(S_t, x_t) + \gamma \mathbb{E}\{V_{t+1}(S_{t+1}(S_t, x_t, W_{t+1}))|S_t\} \big). \tag{9.48}$$

Some students of probability object to this form because it involves writing an expectation as conditional on a random variable instead of conditioning on a filtration as we did in equation (9.45). The problem with conditioning on a filtration is that while it specifies that we are conditioning on the events we can identify at time t, it does not specify precisely what is known, which is what we need when writing Bellman's equation in equation (9.48).

A well-trained probabilist knows that conditioning on the filtration $\mathfrak{F}_t$ and conditioning on the state S_t is the same, since in their view, both are random variables. The state S_t can only take on values in the set $\mathfrak{F}_t$. However, when we are solving problems computationally, the state S_t in (9.48) is viewed as a number (possibly a vector of numbers) which is known at time t.

This highlights a fundamental difference in perspectives between probabilists who think about equations such as Bellman's equation mathematically, and scientists and engineers who are interested in computation. The probabilist can be viewed as someone who is sitting at time 0 in which case S_t is a random variable. The engineer or scientist (or computer program) is sitting at time t where S_t is a number (but it can be any number).

Since probabilists own the theoretical literature on this topic, it is useful to understand how they view a problem. We then offer a bridge from this classical thinking, on which all of probability is based, to the needs of modeling sequential decision problems.

Probabilists represent random phenomena by first defining an outcome ω that is a member of a set Ω that captures every outcome that might happen. Above we said that the probabilist interprets the state S_t as if they are sitting at time 0, which means S_t is a random variable. Actually, the probabilist can be viewed as sitting at time T so she sees the entire set of outcomes Ω (recall that indexing on ω means that you get to see an entire sample path). However, when the probabilist conditions on $\mathfrak{F}_t$, that means she is wearing glasses that allows her to only see the events that can be identified using what is known at time t. So, when conditioning on $\mathfrak{F}_t$, the state S_t (which is really a function of ω) is only allowed to take on outcomes based on events in $\mathfrak{F}_t$.

This notation (which is used universally in probability theory) causes two problems in the context of dynamic programs. First, there are times when we would like to write a decision x_t at time t when following a sample path ω. If we write $x_t(\omega)$, then it means that our decision not only knows that we have been following the sample path ω up to time t, but it also "reveals" the entire sample path, allowing $x_t(\omega)$ to "see" into the future. This causes problems with a popular formulation known as stochastic programming (see chapter 20 for a more thorough discussion).

The second problem, which is more important to us at the moment, is that it complicates taking expectations at some time t, when we are in a state S_t. To a probabilist, the expectation $\mathbb{E}(...)$ *always* represents a summation (or integral) over the event space $(\Omega, \mathfrak{F})$. If we are at time t, probabilists simply replace $\mathfrak{F}$, which captures every possible event that can be defined using the outcomes in Ω, with $\mathfrak{F}_t$, which is the set of all events (that is, the sigma-algebra) that can be defined using just the information available up to time t, given by $W_1, W_2, \ldots, W_t$. This is sometimes written

$$\mathfrak{F}_t = \sigma(W_1, \ldots, W_t),$$

where we would say that $\mathfrak{F}_t$ is the sigma-algebra *generated by* $W_1, \ldots, W_t$. We note that $\mathfrak{F}_t \subseteq \mathfrak{F}_{t+1}$, since the additional information in W_{t+1} allows us to further divide $\mathfrak{F}_t$ into a finer set of events (see section 9.12.1 for an illustration). For this reason, the sequence $\mathfrak{F}_0, \mathfrak{F}_1, \ldots$ is referred to as a *filtration.*

Filtrations fixe the problem of allowing a decision x_t (or $x_t(\omega)$) to "see" into the future. We do this by imposing the condition that "the decision x_t (or policy $X_t^\pi(S_t)$) is $\mathfrak{F}_t$-measurable" which means the decision x_t (or policy $X_t^\pi(S_t)$) uses only the information available up to time t. In practical terms, if there are two outcomes, call them ω_1 and ω_2, in the same event in $\mathfrak{F}_t$, then the decision $x_t(\omega_1)$ and $x_t(\omega_2)$ have to be the same. If they were different, then this could only happen if x_t was being allowed to see events that can only be distinguished at a later point in time.

This language is universally used when writing optimal stopping problems. Imagine that we are interested in selling a stock at the highest price, where prices p_t follow a stochastic process. This problem is often written

$$\max_\tau \mathbb{E} p_\tau x_\tau, \tag{9.49}$$

where $x_t = 1$ if $t = \tau$, and where τ *is an $\mathfrak{F}_t$-measurable random variable*. Note that we are writing τ in (9.49) as a decision variable (which it is) and a random variable (which it is). Once the problem has been stated in this way, it is up to the person solving (9.49) to guarantee that the optimal solution τ^* is "$\mathfrak{F}_t$-measurable."

We pause to note that there is a more practical way of obtaining an "$\mathfrak{F}_t$-measurable function" which is to define a state S_t that is only allowed to depend on the information $W_1, \ldots, W_t$, and then create a function (a policy) $X^\pi(S_t)$ that only depends on the state variable. Saying that a function $X^\pi(S_t)$ depends on the state S_t, and saying that a decision x_t is "$\mathfrak{F}_t$-measurable" is mathematically the same.

When we are solving sequential decision problems, we often have to write expectations when we are at time t, but we have to capture the fact that we are in some state S_t. In the mind of a probabilist, S_t is a random variable, because the probabilist is always sitting at time 0, using concepts such as the filtration $\mathfrak{F}_t$ to create a modified set of events given the information available at time t. With sequential decision problems, it is more useful to assume that we are sitting at time t, in a known state S_t.

9.13 LOOKING FORWARD: DESIGNING POLICIES

We are not quite done with modeling. Chapter 10 addresses the rich area of modeling uncertainty which comes in a number of forms. Once this is done, the rest of the book focuses on designing policies. This material is organized as follows.

Chapter 11 - This chapter describes four fundamental (meta) classes of policies, called policy function approximations (PFAs), cost function approximations (CFAs), policies based on value function approximations (VFAs), and lookahead policies.

Chapter 12 - The simplest class of policies are policy function approximations, which is where we describe a policy as some sort of analytical function (lookup tables, parametric or nonparametric functions)

Chapter 13 - Here we find approximations of cost functions which we then minimize (possibly subject to a set of constraints, which we might also modify).

Chapters 14 - 19 - These chapters develop policies based on value functions. Given the richness of this general approach, we present this material in a series of chapters as follows:

> **Chapter 14** This is the classical material on dynamic programs with discrete states, discrete actions, and randomness that is simple enough that we can take expectations.
>
> **Chapter 15** This chapter covers a series of specialized results that make it possible to solve a dynamic program optimally.
>
> **Chapter 16** This is the first of a series of chapters that present iterative methods for learning approximations of value functions. In this chapter, we introduce a technique we call *backward approximate dynamic programming* since it builds on classical "backward" methods of Markov decision processes presented in chapter 14. The rest of the material on approximate value functions focuses on "forward" methods.
>
> **Chapter 17** We begin with a presentation of methods for approximating value functions using forward methods. In this chapter, the policy is fixed.
>
> **Chapter 18** We build on the tools in chapter 17 but now we use our approximate value functions to define our policy.
>
> **Chapter 19** This chapter focuses on the important special case where the value function is convex in the state variable. This arises in a applications that involve the allocation of resources.

Chapter 20 - The last class of policies optimizes an approximate lookahead model. We deal with two important problem classes: where decisions are discrete (or discretized), and it is possible to enumerate all actions, and where the decision x is a vector, making it impossible to enumerate all actions.

Chapter 21 - We close by addressing the very rich topic of replacing expectation with risk measures. Risk is an important topic when we are dealing with uncertainty, but we then lose the valuable property that the expectation of a sum is the sum of the expectations. Also, we are no longer equally interested in all random outcomes, but rather want to focus our attention on specific outcomes that might cause problems.

9.14 BIBLIOGRAPHIC NOTES

Section 9.2 - Figure 9.1 which describes the mapping from continuous to discrete time was outlined for me by Erhan Cinlar.

Section 9.3 - The definition of states is amazingly confused in the stochastic control literature. The first recognition of the difference between the physical state and the belief state appears to be in Bellman & Kalaba (1959*b*) which used the term "hyperstate" to refer to the belief state. The control literature has long used state to represent a sufficient statistic (see for example Kirk (2004)), representing the information needed to model the system forward in time. For an introduction to partially observable Markov decision processes, see White (1991). An excellent description of the modeling of Markov decision processes from an AI perspective is given in Boutilier et al. (1999), including a very nice discussion of factored representations. See also Guestrin et al. (2003) for an application of the concept of factored state spaces to a Markov decision process.

Section 9.4 - Our notation for decisions represents an effort to bring together the fields of dynamic programming and math programming. We believe this notation was first used in Powell et al. (2001). For a classical treatment of decisions from the perspective of Markov decision processes, see **?**. For examples of decisions from the perspective of the optimal control community, see Kirk (2004) and **?**. For examples of treatments of dynamic programming in economics, see Stokey & R. E. Lucas (1989) and Chow (1997).

Section 9.5 - Our representation of information follows classical styles in the probability literature (see, for example, Chung (1974)). Considerable attention has been given to the topic of supervisory control. An example includes **?**.

Section 9.8 - The energy storage example is taken from **?**.

PROBLEMS

9.1 A traveler needs to traverse the graph shown in figure 9.5 from node 1 to node 11 where the goal is to find the path that minimizes the sum of the costs over the path. To solve this problem we are going to use the deterministic version of Bellman's optimality equation that states

$$V(s) = \min_{a \in \mathcal{A}_s} \big(c(s,a) + V(s'(s,a)) \big) \tag{9.50}$$

where $s'(s,a)$ is the state we transition to when we are in state s and take action $a \in \mathcal{A}_s$. The set $\mathcal{A}_s$ is the set of actions (in this case, traversing over a link) available when we are in state s.

To solve this problem, answer the following questions:

a) Describe an appropriate state variable for this problem (with notation).

b) If the traveler is at node 6 by following the path 1-2-6, what is her state?

c) Find the path that minimizes the sum of the costs on the links traversed by the traveler. Using Bellman's equation (14.2), work backwards from node 11 and find the best

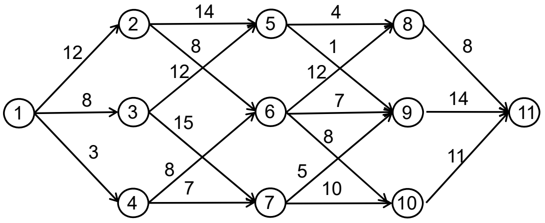


Figure 9.5 Deterministic shortest path problem.

path from each node to node 11, ultimately finding the best path from node 1 to node 11. Show your solution by drawing the graph with the links that fall on an optimal path from some node to node 11 drawn in bold.

9.2 A traveler needs to traverse the graph shown in figure 9.6 from node 1 to node 11, where the goal is to find the path that minimizes the *product* of the costs over the path. To solve this problem, answer the following questions:

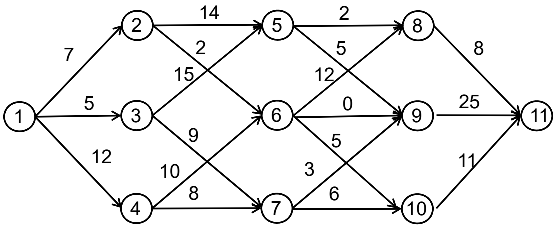


Figure 9.6 A path problem minimizing the product of the costs.

a) Describe an appropriate state variable for this problem (with notation).

b) If the traveler is at node 6 by following the path 1-2-6, what is her state?

c) Using Bellman's equation (14.2), find the path (or paths) that minimizes the product of the costs on the links traversed by the traveler. For each decision point (the nodes in the graph), give the value of the state variable corresponding to the optimal path to that decision point, and the value of being in that state (that is, the cost if we start in that state and then follow the optimal solution).

9.3 A traveler needs to traverse the graph shown in figure 9.7 from node 1 to node 11, where the goal is to find the path that minimizes the *largest* cost of all the links on the path. To solve this problem, answer the following questions:

a) Describe an appropriate state variable for this problem (with notation).

b) If the traveler is at node 6 by following the path 1-2-6, what is her state?

c) Using Bellman's equation (14.2), find the path (or paths) that minimizes the largest costs on the links traversed by the traveler. For each decision point (the nodes in the

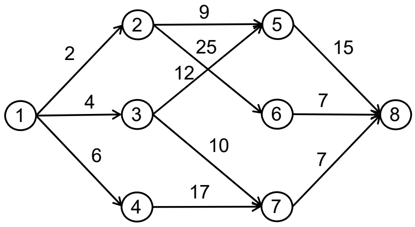


Figure 9.7 A path problem minimizing the product of the costs.

graph), give the value of the state variable corresponding to the optimal path to that decision point, and the value of being in that state (that is, the cost if we start in that state and then follow the optimal solution).

9.4 Repeat exercise 9.3, but this time minimize the *second largest* arc cost on a path.

9.5 Consider our basic newsvendor problem

$$\max_x \mathbb{E}_D F(x, D) = \mathbb{E}_D \big(p \min\{x, D\} - cx\big). \tag{9.51}$$

Show how the following variations of this problem can be modeled using the universal modeling framework:

a) The terminal reward formulation of the basic newsvendor problem.

b) The cumulative reward formulation of the basic newsvendor problem.

c) The asymptotic formulation of the newsvendor problem. What are the differences between the asymptotic formulation and the terminal reward formulation?

9.6 Now consider a dynamic version of our newsvendor problem where a decision x_t is made at time t by solving

$$\max_x \mathbb{E}_D F(x, D) = \mathbb{E}_D \big(p_t \min\{x, D_{t+1}\} - cx\big). \tag{9.52}$$

Assume that the price p_t is independent of prior history.

a) Model the cumulative reward version of the newsvendor problem in (9.52).

b) How does your model change if we are instead solving

$$\max_x \mathbb{E}_D F(x, D) = \mathbb{E}_D \big(p_{t+1} \min\{x, D_{t+1}\} - cx\big). \tag{9.53}$$

where we continue to assume that the price, which is now p_{t+1}, is independent of prior history.

c) How does your model of (9.53) change if

$$p_{t+1} = \theta_0 p_t + \theta_1 p_{t-1} + \varepsilon_{t+1}, \tag{9.54}$$

where ε_{t+1} is a zero-mean noise term, independent of the state of the system.

9.7 We continue the newsvendor problem in exercise 9.6, but now assume that (θ_0, θ_1) in equation (9.54) are unknown. At time t, we have estimates $\bar{\theta}_t = (\bar{\theta}_{t0}, \bar{\theta}_{t1}$. Assume the true θ is now a random variable that follows a multivariate normal distribution with mean $\mathbb{E}_t \theta = \bar{\theta}_t$ which we initialize to

$$\bar{\theta}_0 = \begin{pmatrix} 20 \\ 40 \end{pmatrix},$$

and covariance matrix Σ_t^θ which we initialize to

$$\begin{aligned} \Sigma_0^\theta &= \begin{pmatrix} \sigma_{00}^2 & \sigma_{01}^2 \\ \sigma_{10}^2 & \sigma_{11}^2 \end{pmatrix}. \\ &= \begin{pmatrix} 36 & 16 \\ 16 & 25 \end{pmatrix}. \end{aligned}$$

Drawing on the updating equations in section 3.4.2, give a full model of this problem using a cumulative reward objective function (that is, give the state, decision and exogenous information variables, transition function and objective unction).

9.8 Below is a series of variants of our familiar newsvendor (or inventory) problem. In each, describe the pre- and post-decision states, decision and exogenous information in the form:

$$(S_0, x_0, S_0^x, W_1, S_1, x_1, S_1^x, W_2, \ldots)$$

Specify S_t, S_t^x, x_t and W_t in terms of the variables of the problem.

a) (5 points) The basic newsvendor problem where we wish to find x that solves

$$\max_x \mathbb{E}\{p \min(x, \hat{D}) - cx\} \tag{9.55}$$

where the distribution of $\hat{D}$ is unknown.

b) (3 points) The same as (a), but now we are given a price p_t at time t and asked to solve (9.55) using this information. Note that p_t is unrelated to any prior history or decisions.

c) (3 points) Repeat (b), but now $p_{t+1} = p_t + \hat{p}_{t+1}$.

d) (3 points) Repeat (c), but now leftover inventory is held to the next time period.

e) (3 points) Of the problems above, which (if any) are *not* dynamic programs? Explain.

f) (3 points) Of the problems above, which would be classified as solving state-dependent vs. state-independent functions.

9.9 In this exercise you are going to model an energy storage problem, which is a problem class that arises in many settings (how much cash to keep on hand, how much inventory on a store shelf, how many units of blood to hold, how many milligrams of a drug to keep in a pharmacy, ...). We will begin by describing the problem in English with a smattering of notation. Your job will be to develop it into a formal dynamic model.

Our problem is to decide how much energy to purchase from the electric power grid at a price p_t. Let x_t^{gs} be the amount of power we buy (if $x^{gs} > 0$) or sell (if $x^{gs} < 0$). We then have to decide how much energy to move from storage to meet the demand D_t in a commercial building, where $x_t^{sb} \geq 0$ is the amount we move to the building to meet the demand D_t. Unsatisfied demand is penalized at a price c per unit of energy.

Assume that prices evolve according to a time-series model given by

$$p_{t+1} = \theta_0 p_t + \theta_1 p_{t-1} + \theta_2 p_{t-2} + \varepsilon_{t+1}, \tag{9.56}$$

where ε_{t+1} is a random variable with mean 0 that is independent of the price process. We do not know the coefficients θ_i for $i = 0, 1, 2$, so instead we use estimates $\bar{\theta}_{ti}$. As we observe p_{t+1}, we can update the vector $\bar{\theta}_t$ using the recursive formulas for updating linear models as described in chapter 3, section 3.8 (you will need to review this section to answer parts of this question).

Every time period we are given a forecast $f_{tt'}^D$ of the demand $D_{t'}$ at time t' in the future, where $t' = t, t+1, t+H$. We can think of $f_{tt}^D = D_t$ as the actual demand. We can also think of the forecasts $f_{t+1,t'}^D$ as the "new information" or define a "change in the forecast" $\hat{f}_{t+1,t'}^D$ in which case we would write

$$f_{t+1,t'}^D = f_{tt'}^D + \hat{f}_{t+1,t'}^D.$$

a) What are the elements of the state variable S_t (we suggest filling in the other elements of the model to help identify the information needed in S_t). Define both the pre- and post-decision states.

b) What are the elements of the decision variable x_t? What are the constraints (these are the equations that describe the limits on the decisions). Finally introduce a function $X^\pi(S_t)$ which will be our policy for making decisions to be designed later (but we need it in the objective function below).

c) What are the elements of the exogenous information variable W_{t+1} that become known at time $t+1$ but which were not known at time t.

d) Write out the transition function $S_{t+1} = S^M(S_t, x_t, W_{t+1})$, which is the equations that describe how each element of the state variable S_t evolves over time. There needs to be one equation for each state variable.

e) Write out the objective function by writing:

   The contribution function $C(S_t, x_t)$.

   The objective function where you maximize expected profits over some general set of policies (to be defined later - not in this exercise).

9.10 Patients arrive at a doctor's office, each of whom are described by a vector of attribute $a = (a_1, a_2, \ldots, a_K)$ where a might describe age, gender, height, weight, whether the patient smokes, and so on. Let a^n be the attribute vector describing the nth patient. For each patient, the doctor makes a decision x^n (surgery, drug regimens, rehabilitation), and then observes an outcome y^n for patient n. From y^n, we obtain an updated estimate θ^n for the parameters of a nonlinear model $f(x|\theta)$ that helps us to predict y for other patients.

a) Give the five elements of this decision problem. Be sure to model the state after a patient arrives (this would be the pre-decision state), S^n, after a decision is made (this would be the post-decision state), $S^{x,n}$ and after the outcome of a decision becomes known, $S^{y,n}$.

b) The value of being in a state S^n can be computed using Bellman's equation

$$V^n(S^n) \quad = \quad \max_{x \in \mathcal{X}} \big((C(S^n, x) + E_W\{V^{n+1}(S^{n+1})|S^n, x\}\big). \qquad (9.57)$$

Define the value of being in the state i) after a patient arrives, ii) after a decision is made, and iii) before a patient arrives. Call these $V(S)$, $V^x(S^x)$, and $V^y(S^y)$. Write $V(S^n)$ as a function of $V^x(S^{x,n})$ and write $V^x(S^{x,n})$ as a function of $V^y(S^{y,n})$.

9.11 Consider the problem of controlling the amount of cash a mutual fund keeps on hand. Let R_t be the cash on hand at time t. Let $\hat{R}_{t+1}$ be the net deposits (if $\hat{R}_{t+1} > 0$) or withdrawals (if $\hat{R}_{t+1} < 0$), where we assume that $\hat{R}_{t+1}$ is independent of $\hat{R}_t$. Let M_t be the stock market index at time t, where the evolution of the stock market is given by $M_{t+1} = M_t + \hat{M}_{t+1}$ where $\hat{M}_{t+1}$ is independent of M_t. Let x_t be the amount of money moved from the stock market into cash ($x_t > 0$) or from cash into the stock market ($x_t < 0$).

a) Give a complete model of the problem, including both pre-decision and post-decision state variables.

b) Suggest a simple parametric policy function approximation, and give the objective function as an online learning problem.

9.12 A college student must plan what courses she takes over each of eight semesters. To graduate, she needs 34 total courses, while taking no more than five and no less than three courses in any semester. She also needs two language courses, one science course, eight departmental courses in her major and two math courses.

(a) Formulate the state variable for this problem in the most compact way possible.

(b) Give the transition function for our college student assuming that she successfully passes any course she takes. You will need to introduce variables representing her decisions.

(c) Give the transition function for our college student, but now allow for the random outcome that she may not pass every course.

9.13 Assume that we have N discrete resources to manage, where R_a is the number of resources of type $a \in \mathcal{A}$ and $N = \sum_{a \in \mathcal{A}} R_a$. Let $\mathcal{R}$ be the set of possible values of the vector R. Show that

$$|\mathcal{R}| = \begin{pmatrix} N + |\mathcal{A}| - 1 \\ |\mathcal{A}| - 1 \end{pmatrix},$$

where

$$\begin{pmatrix} X \\ Y \end{pmatrix} = \frac{X!}{Y!(X-Y)!}$$

is the number of combinations of X items taken Y at a time.

9.14 A broker is working in thinly traded stocks. He must make sure that he does not buy or sell in quantities that would move the price and he feels that if he works in quantities that are no more than 10 percent of the average sales volume, he should be safe. He tracks the average sales volume of a particular stock over time. Let $\hat{v}_t$ be the sales volume on day t, and assume that he estimates the average demand f_t using $f_t = (1-\alpha)f_{t-1} + \alpha\hat{v}_t$. He then uses f_t as his estimate of the sales volume for the next day. Assuming he started tracking demands on day $t = 1$, what information would constitute his state variable?

9.15 How would your previous answer change if our broker used a 10-day moving average to estimate his demand? That is, he would use $f_t = 0.10\sum_{i=1}^{10} \hat{v}_{t-i+1}$ as his estimate of the demand.

9.16 The pharmaceutical industry spends millions managing a sales force to push the industry's latest and greatest drugs. Assume one of these salesmen must move between a set $\mathcal{I}$ of customers in his district. He decides which customer to visit next only after he completes a visit. For this exercise, assume that his decision does not depend on his prior history of visits (that is, he may return to a customer he has visited previously). Let S_n be his state immediately after completing his n^{th} visit that day.

(a) Assume that it takes exactly one time period to get from any customer to any other customer. Write out the definition of a state variable, and argue that his state is only his current location.

(b) Now assume that τ_{ij} is the (deterministic and integer) time required to move from location i to location j. What is the state of our salesman at any time t? Be sure to consider both the possibility that he is at a location (having just finished with a customer) or between locations.

(c) Finally assume that the travel time τ_{ij} follows a discrete uniform distribution between a_{ij} and b_{ij} (where a_{ij} and b_{ij} are integers)?

9.17 Consider a simple asset acquisition problem where x_t is the quantity purchased at the end of time period t to be used during time interval $t+1$. Let D_t be the demand for the assets during time interval t. Let R_t be the pre-decision state variable (the amount on hand before you have ordered x_t) and R_t^x be the post-decision state variable.

(a) Write the transition function so that R_{t+1} is a function of $R_t, x_t,$ and D_{t+1}.

(b) Write the transition function so that R_t^x is a function of $R_{t-1}^x, D_t,$ and x_t.

(c) Write R_t^x as a function of R_t, and write R_{t+1} as a function of R_t^x.

9.18 As a buyer for an orange juice products company, you are responsible for buying futures for frozen concentrate. Let $x_{tt'}$ be the number of futures you purchase in year t that can be exercised during year t'.

(a) What is your state variable in year t?

(b) Write out the transition function.

9.19 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 $p_d = P(D_t = d)$. 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 x_t at the end of period t that can be used to replenish the inventory in period $t+1$. Give the transition function that relates R_{t+1} to R_t.

9.20 Many problems involve the movement of resources over networks. The definition of the state of a single resource, however, can be complicated by different assumptions for the probability distribution for the time required to traverse a link. For each example below, give the state of the resource:

(a) You have a deterministic, static network, and you want to find the shortest path from an origin node q to a destination node r. There is a known cost c_{ij} for traversing each link (i, j).

(b) Next assume that the cost c_{ij} is a random variable with an unknown distribution. Each time you traverse a link (i, j), you observe the cost $\hat{c}_{ij}$, which allows you to update your estimate $\bar{c}_{ij}$ of the mean of c_{ij}.

(c) Finally assume that when the traveler arrives at node i he sees $\hat{c}_{ij}$ for each link (i, j) out of node i.

(d) A taxicab is moving people in a set of cities $\mathcal{C}$. After dropping a passenger off at city i, the dispatcher may have to decide to reposition the cab from i to j, $(i, j) \in \mathcal{C}$. The travel time from i to j is τ_{ij}, which is a random variable with a discrete uniform distribution (that is, the probability that $\tau_{ij} = t$ is $1/T$, for $t = 1, 2, \ldots, T$). Assume that the travel time is known before the trip starts.

(e) Same as (d), but now the travel times are random with a geometric distribution (that is, the probability that $\tau_{ij} = t$ is $(1-\theta)\theta^{t-1}$, for $t = 1, 2, 3, \ldots$).

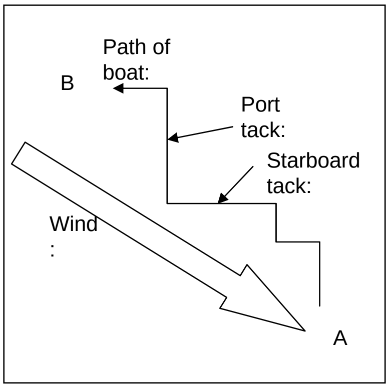
9.21 In the figure below, a sailboat is making its way upwind from point A to point B. To do this, the sailboat must tack, whereby it sails generally at a 45 degree angle to the wind. The problem is that the angle of the wind tends to shift randomly over time. The skipper decides to check the angle of the wind each minute and must decide whether the boat should be on port or starboard tack. Note that the proper decision must consider the current location of the boat, which we may indicate by an (x, y) coordinate.

(a) Formulate the problem as a dynamic program. Carefully define the state variable, decision variable, exogenous information and the contribution function.

(b) Use δ to discretize any continuous variables (in practice, you might choose different levels of discretization for each variable, but we are going to keep it simple). In terms of δ, give the size of the state space, the number of exogenous outcomes (in a single time period) and the action space. If you need an upper bound on a variable (e.g. wind speed), simply define an appropriate variable and express your answer in terms of this variable. All your answers should be expressed algebraically.

(c) Using a maximum wind speed of 30 miles per hour and $\delta = .1$, compute the size of your state, outcome and action spaces.

9.22 Implement your model from exercise 9.21 as a Markov decision process, and solve it using the techniques of 14 (section 14.2). Choose a value of δ that makes your program



computationally reasonable (run times under 10 minutes). Let $\bar{\delta}$ be the smallest value of δ that produces a run time (for your computer) of under 10 minutes, and compare your solution (in terms of the total contribution) for $\delta = \bar{\delta}^N$ for $N = 2, 4, 8, 16$. Evaluate the quality of the solution by simulating 1000 iterations using the value functions obtained using backward dynamic programming. Plot your average contribution function as a function of δ.

9.23 What is the difference between the *history* of a process, and the state of a process?

9.24 As the purchasing manager for a major citrus juice company, you have the responsibility of maintaining sufficient reserves of oranges for sale or conversion to orange juice products. Let x_{ti} be the amount of oranges that you decide to purchase from supplier i in week t to be used in week $t + 1$. Each week, you can purchase up to $\hat{q}_{ti}$ oranges (that is, $x_{ti} \leq \hat{q}_{ti}$) at a price $\hat{p}_{ti}$ from supplier $i \in \mathcal{I}$, where the price/quantity pairs $(\hat{p}_{ti}, \hat{q}_{ti})_{i \in \mathcal{I}}$ fluctuate from week to week. Let s_0 be your total initial inventory of oranges, and let D_t be the number of oranges that the company needs for production during week t (this is our demand). If we are unable to meet demand, the company must purchase additional oranges on the spot market at a spot price $\hat{p}_{ti}^{spot}$.

(a) What is the exogenous stochastic process for this system?

(b) What are the decisions you can make to influence the system?

(c) What would be the state variable for your problem?

(d) Write out the transition equations.

(e) What is the one-period contribution function?

(f) Propose a reasonable structure for a decision rule for this problem, and call it X^π. Your decision rule should be in the form of a function that determines how much to purchase in period t.

(g) Carefully and precisely, write out the objective function for this problem in terms of the exogenous stochastic process. Clearly identify what you are optimizing over.

(h) For your decision rule, what do we mean by the space of policies?

9.25 Customers call in to a service center according to a (nonstationary) Poisson process. Let $\mathcal{E}$ be the set of events representing phone calls, where $t_e, e \in \mathcal{E}$ is the time that the call is made. Each customer makes a request that will require time τ_e to complete and will pay a reward r_e to the service center. The calls are initially handled by a receptionist who determines τ_e and r_e. The service center does not have to handle all calls and obviously favors calls with a high ratio of reward per time unit required (r_e/τ_e). For this reason, the company adopts a policy that the call will be refused if $(r_e/\tau_e) < \gamma$. If the call is accepted, it is placed in a queue to wait for one of the available service representatives. Assume that the probability law driving the process is known, where we would like to find the right value of γ.

(a) This process is driven by an underlying exogenous stochastic process with element $\omega \in \Omega$. What is an instance of ω?

(b) What are the decision epochs?

(c) What is the state variable for this system? What is the transition function?

(d) What is the action space for this system?

(e) Give the one-period reward function.

(f) Give a full statement of the objective function that defines the Markov decision process. Clearly define the probability space over which the expectation is defined, and what you are optimizing over.

9.26 A major oil company is looking to build up its storage tank reserves, anticipating a surge in prices. It can acquire 20 million barrels of oil, and it would like to purchase this quantity over the next 10 weeks (starting in week 1). At the beginning of the week, the company contacts its usual sources, and each source $j \in \mathcal{J}$ is willing to provide $\hat{q}_{tj}$ million barrels at a price $\hat{p}_{tj}$. The price/quantity pairs $(\hat{p}_{tj}, \hat{q}_{tj})$ fluctuate from week to week. The company would like to purchase (in discrete units of millions of barrels) x_{tj} million barrels (where x_{tj} is discrete) from source j in week $t \in \{1, 2, \ldots, 10\}$. Your goal is to acquire 20 million barrels while spending the least amount possible.

(a) What is the exogenous stochastic process for this system?

(b) What would be the state variable for your problem? Give an equation(s) for the system dynamics.

(c) Propose a structure for a decision rule for this problem and call it X^π.

(d) For your decision rule, what do we mean by the space of policies? Give examples of two different decision rules.

(e) Write out the objective function for this problem using an expectation over the exogenous stochastic process.

(f) You are given a budget of $300 million to purchase the oil, but you absolutely must end up with 20 million barrels at the end of the 10 weeks. If you exceed the initial budget of $300 million, you may get additional funds, but each additional $1 million will cost you $1.5 million. How does this affect your formulation of the problem?

9.27 You own a mutual fund where at the end of each week t you must decide whether to sell the asset or hold it for an additional week. Let $\hat{r}_t$ be the one-week return (e.g. $\hat{r}_t = 1.05$ means the asset gained five percent in the previous week), and let p_t be the price of the asset if you were to sell it in week t (so $p_{t+1} = p_t\hat{r}_{t+1}$). We assume that the returns $\hat{r}_t$ are independent and identically distributed. You are investing this asset for eventual use in your college education, which will occur in 100 periods. If you sell the asset at the end of time period t, then it will earn a money market rate q for each time period until time period 100, at which point you need the cash to pay for college.

(a) What is the state space for our problem?

(b) What is the action space?

(c) What is the exogenous stochastic process that drives this system? Give a five time period example. What is the history of this process at time t?

(d) You adopt a policy that you will sell if the asset falls below a price $\bar{p}$ (which we are requiring to be independent of time). Given this policy, write out the objective function for the problem. Clearly identify exactly what you are optimizing over.

CHAPTER 10

UNCERTAINTY MODELING

We cannot find an effective policy unless we are modeling the problem properly. In the realm of stochastic optimization, this means accurately modeling uncertainty. The importance of modeling uncertainty has been underrepresented in the stochastic optimization literature, although practitioners working on real problems have long been aware of the challenges of modeling uncertainty.

Fortunately, there is a substantial body of research focused on the modeling of uncertainty and stochastic processes that has evolved in the communities working on Monte Carlo simulation and uncertainty quantification. We use uncertainty modeling as the broader term that describes the process of identifying and modeling uncertainty, while simulation refers to the vast array of tools that break down complex stochastic processes using the power of Monte Carlo simulation.

It helps to remind ourselves of the two information processes that drive any sequential stochastic optimization problem: decisions, and exogenous information. Assume that we can pick some policy $X_t^\pi(S_t)$. We need to be able to simulate a sample realization of the policy, which will look like

$$S_0 \rightarrow x_0 = X_0^\pi(S_0) \rightarrow W_1 \rightarrow S_1 \rightarrow x_1 = X_1^\pi(S_1) \rightarrow W_2 \rightarrow S_3 \rightarrow$$

Given our policy, this simulation assumes that we have access to a transition function

$$S_{t+1} = S^M(S_t, X_t^\pi(S_t), W_{t+1}). \tag{10.1}$$

We can execute equation (10.1) if we have access to the following:

S_0 = The initial state - This is where we place information about initial estimates (or priors) of parameters, as well as assumptions about probability distributions and functions.

W_t = Exogenous information that enters our system for the first time between $t-1$ and t.

In this chapter, we focus on the often challenging problem of simulating the exogenous sequence $(W_t)_{t=0}^T$. We assume that the initial state S_0 is given, but recognize that it may include a probabilistic belief about unknown and unobservable parameters. The process of converting the characteristics of a stochastic process into a mathematical model is broadly known as *uncertainty quantification.* Since it is easy to overlook sources of uncertainty when building a model, we place considerable attention on identifying the different sources of uncertainty that we have encountered in our applied work, keeping in mind that S_0 and W_t are the only variables our modeling framework provides for representing uncertainty.

After reviewing different types of uncertainty, we then provide a basic introduction to a powerful set of techniques known as Monte Carlo simulation, which allows us to replicate stochastic processes on the computer. Given the rich array of different types of stochastic processes, our discussion here provides little more than a taste of the tools that are available to replicate stochastic processes.

10.1 TYPES OF UNCERTAINTY

Uncertainty arises in different forms. Some of the major forms that we have encountered are

- Observational errors - This arises from uncertainty in observing or measuring the state of the system. Observational errors arise when we have unknown state variables that cannot be observed directly (and accurately).

- Exogenous uncertainty - This describes the exogenous arrival of information to the system, which might be weather, demands, prices, the response of a patient to medication or the reaction of the market to a product.

- Prognostic uncertainty - We often have access to a forecast $f_{tt'}^W$ of the information $W_{t'}$. Prognostic uncertainty captures the deviation of the actual $W_{t'}$ from the forecast $f_{tt'}^W$. If we think of $W_t = f_{tt}^W$ as the actual value of W_t, then we can think of the realization of W_t (the exogenous information described above) as just an update to a forecast.

- Inferential (or diagnostic) uncertainty - Inferential uncertainty arises when we use observations (from field or physical measurements, or computer simulations) to draw inferences about another set of parameters. It arises from our lack of understanding of the precise properties or behavior of a system, which introduces errors in our ability to estimate parameters, partly from noise in the observations, and partly from errors in our modeling of the underlying system.

- Experimental variability - Sometimes equated with observational uncertainty, experimental variability refers to differences between the results of experiments run under similar conditions. An experiment might be a computer simulation, a laboratory experiment or a field implementation. Even if we can perfectly measure the results of an experiment, there is variation from one experiment to the next.

- Model uncertainty - We may not know the structure of the transition function $S_{t+1} = S^M(S_t, x_t, W_{t+1})$, or the parameters that are imbedded in the function. Model uncertainty is often attributed to the transition function, but it may also apply to the model of the stochastic process W_t since we often do not know the precise structure.

- Transitional uncertainty - This arises when we have a perfect model of how a system should evolve, but exogenous shocks (wind buffeting an aircraft, rainfall affecting reservoir levels) can introduce uncertainty in how an otherwise deterministic system will evolve.

- Control/implementation uncertainty - This is where we choose a control u_t (such as a temperature or speed), but what happens is $\hat{u}_t = u_t + \delta u_t$ where δu_t is a random perturbation.

- Communication errors and biases - Communication from an agent q about his state S_{qt} to an agent q' where errors may introduced, either accidentally or purposely.

- Algorithmic instability - Very minor changes in the input data for a problem, or small adjustments in parameters guiding an algorithm (which exist in virtually all algorithms), can completely change the path of the algorithm, introducing variability in the results.

- Goal uncertainty - Uncertainty in the desired goal of a solution, as might arise when a single model has to produce results acceptable to different people or users.

- Political/regulatory uncertainty - Uncertainty about taxes, rules and requirements that affect costs and constraints (for example, tax energy credits, automotive mileage standards). These can be viewed as a form of systematic uncertainty, but this is a particularly important source of uncertainty with its own behaviors.

Below we provide more detailed discussions of each type of uncertainty. One challenge is modeling each source of uncertainty, since we have only two mechanisms for introducing exogenous information into our model: the initial state S_0, and the exogenous information process $W_1, W_2, \ldots$. Thus, the different types of uncertainty may look similar mathematically, but it is important to characterize the mechanisms by which uncertainty enters our model.

10.1.1 Observational errors

Observational (or measurement) uncertainty reflects errors in our ability to observe (or measure) the state of the system directly. Some examples include:

■ **EXAMPLE 10.1**

Different people may measure the gases in the oil of a high-voltage transformer, producing different measurements (possibly due to variations in equipment, the temperature at which the transformer was observed, or variations in the oil surrounding the coils).

■ **EXAMPLE 10.2**

The Center for Disease Control estimates the number of mosquitoes carrying a disease by setting traps and counting how many mosquitoes are caught that are found with the disease. From day to day the number of infected mosquitoes that are caught can vary considerably.

■ **EXAMPLE 10.3**

A company may be selling a product at a price p_t which is being varied to find the best price. However, the sales (at a fixed price) will be random from one time period to the next.

■ **EXAMPLE 10.4**

Different doctors, seeing the same patient for the first time, may elicit different information about the characteristics of the patient.

Partially observable systems arise in any application where we cannot directly observe parameters. A simple example arises in pricing, where we may feel that demand varies linearly with price according to

$$D(p) = \theta_0 - \theta_1 p.$$

At time t, our best estimate of the demand function is given by

$$D(p) = \bar{\theta}_0 - \bar{\theta}_1 p.$$

We observe sales, which would be given by

$$\hat{D}_{t+1} = \theta_0 - \theta_1 p_t + \varepsilon_{t+1}.$$

We do not know (θ_0, θ_1), but we can use observations to create updated estimates. If $(\bar{\theta}_{t0}, \bar{\theta}_{t1})$ is our estimate as of time t, we can use our observation $\hat{D}_{t+1}$ of sales between t and $t+1$ to obtain updated estimates $(\bar{\theta}_{t+1,0}, \bar{\theta}_{t+1,1})$. In this model, we would view $\bar{\theta}_t = (\bar{\theta}_{t0}, \bar{\theta}_{t1})$ as our state variable, which is our estimate of the static parameter θ. Since θ is a fixed parameter, we do not include it in the state variable, but rather treat it as a *latent variable*.

The presence of states that cannot be perfectly observable gives rise to what are widely known as *partially observable Markov decision processes*, or POMDP's. To model this, let $\check{S}_t$ be the true (but possibly unobservable) state of the system at time t, while S_t is the observable state. One way of writing our dynamics might be

$$S_{t+1} = \check{S}^M(\check{S}_t, a_t) + \varepsilon_{t+1},$$

which captures our inability to directly observe $\check{S}_t$. These systems are most often motivated by problems such as those in engineering where we cannot directly observe the state of charge of a battery, the location and velocity of an aircraft, or the number of truck trailers sitting at a terminal (terminal managers tend to hide trailers to keep up their inventories).

We can represent our unobservable state as a probability distribution. This might be a continuous distribution (perhaps the normal or multivariate normal distribution), or perhaps more simply as a discrete distribution where q_{ti}^k is the probability that the state variable S_{ti} takes on outcome k (or perhaps a parameter θ^k) at time t. Then, the vector $q_{ti} = (q_{ti}^k)_k$ is the distribution capturing our belief about the unobservable state. We then include q_t (for each uncertain state dimension) as part of our state variable (this is where our belief state comes in).

10.1.2 Exogenous uncertainty

Exogenous uncertainty represent the information that we typically model through the process W_t represent new information about supplies and demands, costs and prices, and physical parameters that can appear in either the objective function or constraints. Exogenous uncertainty can arise in different styles, including:

- Fine time-scale uncertainty - Sometimes referred to as *aleatoric uncertainty*, fine time-scale uncertainty refers to uncertainty that varies from time-step to time-step which is assumed to reflect the dynamics of the problem. Whether a time step is minutes, hours, days or weeks, fine time-scale uncertainty means that information from one time-step to the next is either uncorrelated, or where correlations drop off fairly quickly.

- Coarse time-scale uncertainty - Referred to in different settings as systematic uncertainty or *epistemic* uncertainty (popular in the medical community), coarse time-scale uncertainty reflects uncertainty in an environment which occurs over long time scales. This might reflects new technology, changes in market patterns, the introduction of a new disease, or an unobserved fault in machinery for a process.

- Distributional uncertainty - If we represent the exogenous information W_t, or the initial state S_0, as a probability distribution, there may be uncertainty in either the type of distribution or the parameters of a distribution.

- Adversarial uncertainty - The exogenous information process $W_1, \ldots, W_T$ may come from an adversary who is choosing W_t in a way to make us perform poorly. We cannot be sure how the adversary may behave.

10.1.3 Prognostic uncertainty

Prognostic uncertainty reflects errors in our ability to forecast activities in the future. Typically these are written as $f_{tt'}$ to represent the forecast of some quantity at time t', given what we know at time t (represented by our state variable S_t). Examples include:

■ **EXAMPLE 10.1**

A company may create a forecast of demand D_t for its product. If $f_{tt'}^D$ is the forecast of the demand $D_{t'}$ given what we know at time t, then the difference between $f_{tt'}^D$ and $D_{t'}$ is the uncertainty in our forecast.

■ **EXAMPLE 10.2**

A utility is interested in forecasting the price of electricity 10 years from now. Electricity prices are well approximated by the intersection of the load (the amount of electricity needed at a point in time) and the "supply stack" which is the cost of energy as a function of the total supply (typically an increasing function). The supply stack reflects the cost of different fuels (nuclear, coal, natural gas). We have to forecast the prices of these different sources (one form of uncertainty) along with the load (a different form of uncertainty).

■ **EXAMPLE 10.3**

We might be interested in forecasting energy from wind $E^W_{t'}$ at time t. This might require that we first generate a meteorological forecast of weather systems (high and low pressure systems), as well as capturing the movement of the atmosphere (wind speed and direction).

If $W_{t'}$ is some form of random information in the future, we might be able to create a forecast $f^W_{tt'}$ using what we know at time t. We typically assume that our forecasts are unbiased, which means we can write

$$f^W_{tt'} = \mathbb{E}\{W_{t'}|S_t\}.$$

Forecasts can come from two sources. An *endogenous forecast* is obtained from a model that is created endogenously from data. For example, we might be forecasting demand using the model

$$f^D_{tt'} = \theta_{t0} + \theta_{t1}(t' - t).$$

Now assume we observe the demand D_{t+1}. We might use any of a range of algorithms to update our parameter estimates to obtain

$$f^D_{t+1,t'} = \theta_{t+1,0} + \theta_{t+1,1}(t' - (t+1)).$$

The parameter vector θ_t can be updated recursively from observations W_{t+1}. If θ_t is our current estimate of $(\theta_{t0}, \theta_{t1})$, let Σ_t be our estimate of the covariance between the random variables θ_0 and θ_1 (these are the true values of the parameters). Let $\beta^W = 1/(\sigma^2_W)$ be the precision of an observation W_{t+1} (the precision is the inverse of the variance), and assume we can form the precision matrix given by $B_t = [(X_t)^T X_t]^{-1}$, where X_t is a matrix where each row consists of the vector of independent variables (in the case of our demand example, the design variables for time t would be $x_t = (1 \;\; p_t)$). We can update θ_t and Σ_t (or B_t) recursively using

$$\theta_{t+1} = \theta_t - \frac{1}{\gamma_{t+1}} B_t x_{t+1} \varepsilon_{t+1}, \tag{10.2}$$

where ε_{t+1} is the error given by

$$\varepsilon_{t+1} = W_{t+1} - \theta_t x_t. \tag{10.3}$$

The matrix $B_{t+1} = [(X_{t+1})^T X_{t+1}]^{-1}$. This can be updated recursively without computing an explicit inverse using

$$B_{t+1} = B_t - \frac{1}{\gamma_{t+1}}(B_t x_{t+1}(x_{t+1})^T B_t). \tag{10.4}$$

The parameter γ_{t+1} is a scalar computed using

$$\gamma_{t+1} \quad = \quad 1 + (x_{t+1})^T B_t x_{t+1}. \tag{10.5}$$

Note that if we multiply (10.4) through by σ_ϵ^2 we obtain

$$\Sigma_{t+1}^\theta = \Sigma_t^\theta - \frac{1}{\gamma_{t+1}}(\Sigma_t^\theta x_{t+1}(x_{t+1})^T \Sigma_t^\theta), \tag{10.6}$$

where we scale γ_{t+1} by σ_ϵ^2, giving us

$$\gamma_{t+1} \quad = \quad \sigma_\epsilon^2 + (x_{t+1})^T \Sigma_t^\theta x_{t+1}. \tag{10.7}$$

Equations (10.2)-(10.7) represent the transition function for updating θ_t.

The second source of a forecast is exogenous, where the forecast might be supplied by a vendor. In this case, we might view the updated set of forecasts $(f_{tt'})_{t' \geq t}$ as exogenous information. Alternatively, we could think of the change in forecasts as the exogenous information. If we let $\hat{f}_{t+1,t'}$ as the change between t and $t+1$ in the forecast for activities at time t', we would then write

$$f_{t+1,t'} = f_{tt'} + \hat{f}_{t+1,t'}.$$

From a modeling perspective, these forecasts differ in terms of how they are represented in the state variable. In the case of our endogenous forecast, the state variable would be captured by (θ_t, Σ_t), with the corresponding transition equations given by (10.2)-(10.7). With our exogenous forecast, the state variable would be simply $(f_{tt'})_{t'=t}^T$.

Regardless of whether the forecast is exogenous or endogenous, the new information (the exogenous observation or the updated forecast) would be modeled as a part of the exogenous information process W_t.

10.1.4 Inferential (or diagnostic) uncertainty

It is often the case that we cannot directly observe a parameter. Instead, we have to use (possibly imperfect) observations of one or more parameters to infer variables or parameters that we cannot directly observe. Some examples include:

■ **EXAMPLE 10.1**

We might not be able to directly observe the presence of heart disease, but we may use blood pressure as an indicator. Measuring blood pressure introduces observational error, but there is also error in making the inference that a patient suffers from heart disease from blood pressure alone.

■ **EXAMPLE 10.2**

We observe (possibly imperfectly) the sales of a product. From these sales we wish to estimate the elasticity of demand with respect to price.

■ **EXAMPLE 10.3**

Power companies generally do not know the precise location of a tree falling that creates a power outage. Rather, a falling tree can create a short that will trip a circuit breaker higher in the circuit, producing many outages, including to customers who may be far from the fallen tree. Diagnostic uncertainty refers to errors in our ability to precisely describe where a tree might have fallen purely from phone calls.

■ **EXAMPLE 10.4**

Sensors may detect an increase in carbon monoxide in the exhaust of a car. This information may indicate several possible causes, such as an aging catalytic converter, the improper timing of the cylinders, or an incorrect air-fuel mixture (which might hint at a problem in a different sensor).

Inferential uncertainty can be described as uncertainty in the parameters of a model. In our example involving the detection of carbon monoxide, we might use this information to update the probability that the real cause is due to each of three or four different mechanical problems. This would represent an instance of using a (possibly noisy) observation to update a lookup table model of where failures are located. By contrast, when we use sales data to update our demand elasticity, that would be an example of using noisy observational data to update a parametric model.

In some settings the term *diagnostic uncertainty* is used instead of inferential uncertainty. We feel that this term reflects the context of identifying a problem (a failed component, presence of a disease) that we are not able to observe directly. However, both inferential uncertainty and diagnostic uncertainty reflect uncertainty in parameters that have been estimated (inferred) from indirect observations.

Inferential uncertainty is a form of derived uncertainty that arises when we estimate a parameter $\bar{\theta}$ from data (simulated or observed). The raw uncertainty is contained in the sequence W_t (or W^n). We then have to derive the distribution of our estimate $\bar{\theta}$ resulting from the exogenous noise, which we contain in our belief state B_t.

10.1.5 Experimental variability

Experimental variability reflects changes in the results of experiments run under the same conditions. Experimental settings include

Laboratory experiments - We include here physical experiments run in a laboratory setting, encompassing chemical, biological, mechanical and even human testing.

Numerical simulations - Large simulators describing complex physical systems, ranging from models of businesses to models of physical processes, can exhibit variability from one run to the next, often reflecting minor variations in input data and parameters.

Field testing - This can range from observing sales of a product to testing of new drugs.

Experimental uncertainty arises from possibly minor variations in the dynamics of a system (simulated or physical) which introduce variability when running experiments. Experimental uncertainty typically reflects our inability to perfectly estimate parameters that drive the system, or errors in our ability to understand (or model) the system.

Some sources equate observational and experimental uncertainty, and often they are handled in the same way. However, we feel it is useful to distinguish between pure measurement (observational) errors, which might be reduced with better technology, and experimental errors, which have more to do with the process and which are not reduced through better measurement technologies.

Experimental noise might be attributed as a byproduct of the exogenous information process W_t. For example, for a given policy $X^\pi(S_t)$, an experiment might consist of evaluating

$$\hat{F}^\pi = F^\pi(\omega) = \sum_{t=0}^{T} C(S_t(\omega), X^\pi(S_t(\omega))).$$

Here, the noise is due to the variation in W_t. However, imagine that we are running a series of experiments. Let $\hat{F}^n(\theta^n)$ be the observation of the outcome of an experiment run with parameters $\theta = \theta^n$. Let $f(\theta) = \mathbb{E}\hat{F}^n(\theta^n)$ be the exact (but unobservable) value of running the experiment with parameter setting θ. We can write

$$\hat{F}^n(\theta^n) = f(\theta^n) + \varepsilon^n.$$

In this case, the sequence ε^n would be the exogenous information information W^n.

10.1.6 Model uncertainty

Model uncertainty comes in two forms. The first is errors in estimates of parameters of a parametric model. If we are estimating these parameters over time from observations, we would refer to this as inferential uncertainty. But now imagine that we characterize our model using a set of fixed parameters that are not being updated. We are not estimating these parameters over time, but rather we are just using assumed values which are uncertain.

The second is errors in the structure of the model itself (economists refer to this as *specification errors*). Some examples include:

■ **EXAMPLE 10.1**

We may approximate demand as a function of price as a linear function, a logistics curve, or a quadratic function. We will use observational data to estimate the parameters of each function, but we may not directly address the errors introduced by assuming a particular type of function.

■ **EXAMPLE 10.2**

We may describe the diffusion of chemicals in a liquid using a first-order set of differential equations, which we fit to observational data. But the real process may be better described by a second (or higher) order set of differential equations. Our first-order model may be at best a good local approximation.

■ **EXAMPLE 10.3**

Grid operators often model the supply curve of a generator using a convex function, which is easier to solve. However, a more detailed model might capture complex relationships that reflect the fact that costs may rise in steps as different components of the generator come on (e.g. heat recovery).

Model uncertainty for dynamic problems can be found in three different parts of the model:

- Costs or rewards - Measuring the cost of a grid outage on the community may require estimating the impact of a loss of power on homes and businesses.
- Constraints - Constraints can often be written in the form $A_t x_t = R_t$. There are many applications where dynamic uncertainty enters through the right hand side R_t; this is how we would model the supply or demand of blood which would be a more typical form of dynamic uncertainty. Model uncertainty often arises in the matrix A_t, which is where we might capture the assumed speed of an aircraft, or the efficiency of a manufacturing process.
- Dynamics - This is where we are uncertain about the function $S^M(S_t, a_t, W_{t+1})$ which describes how the system evolves over time.

The transition function $S^M(\cdot)$ captures all the physics of a problem, and there are many problems where we simply do not understand the physics. For example, we might be trying to explain how a person or market might respond to a price, or how global warming might respond to a change in CO2 concentrations. Some policies make decisions using nothing more than the current state, allowing them to be used in settings where the underlying dynamics have not been modeled. By contrast, an entire class of policies based on lookahead models (which we cover in chapter 20) depend on at least an approximate model of the problem. See 9.6.2 for a more thorough discussion of model-free dynamic programming.

Whether we are dealing with costs, constraints or the dynamics, our model can be described in terms of the choice of the model structure, and any parameters that characterize the model. Let $m \in \mathcal{M}$ represent the structure of the model, and let $\theta \in \Theta^m$ be the parameters that characterize a model with structure m. As a general rule, the model structure m is fixed in advance (for example, we might assume that a particular relationship is linear) but with uncertain parameters. However, this is not always the case, and we may associate a prior q_0^m that gives the probability that we believe that model m is correct. Similarly, we might start with an initial estimate θ_0^m for the parameter vector θ^m. We might even assume that we start by assuming that θ^m is described by a multivariate normal distribution with mean θ_0^m and covariance matrix Σ_0^m.

As we might expect, prior information about the model (whether it is the probability q_0 that a type of model is correct, or the prior distribution on θ^m) is communicated through the initial state S_0. If this belief is updated over time, then this would also be part of the dynamic state S_t.

10.1.7 Transitional uncertainty

There are many problems where the dynamics of the system are modeled deterministically. This is often the case in engineering applications where we apply a control u_t (such as a force) to a dynamic system. Simple physics might describe how the control affects our system, which we would then write

$$S_{t+1} = S^M(S_t, u_t).$$

However, exogenous noise might interfere with these dynamics. For example, we might be predicting the speed and location of an aircraft after applying forces u_t. Variations in the atmosphere might interfere with our equations, so we introduce a noise term ε_{t+1}, giving us

$$S_{t+1} = S^M(S_t, u_t) + \varepsilon_{t+1}.$$

We note that despite the noise, we assume that we can observe (measure) the state perfectly.

10.1.8 Control/implementation uncertainty

There are many problems where we cannot precisely control a process. Some examples include:

■ **EXAMPLE 10.1**

An experimentalist has requested that a rate be fed a diet with x_t grams of fat. However, variability in the preparation of the meals, and the choice of the rat of what to eat, introduces variability in the amount of fat that is consumed.

■ **EXAMPLE 10.2**

A publisher chooses to sell a book at a wholesale price p_t^W at time t and then observes sales. However, the publisher has no control over the retail price offered to the purchasing public.

■ **EXAMPLE 10.3**

The operator of a power grid may request that a generator come online and generate x_t megawatts of power. However, this may not happen either because of a technical malfunction or human implementation errors.

Control uncertainty is widely overlooked in the dynamic programming literature, but is well known in the econometrics community as the “errors in variable” model.

We might model errors in the implementation of a decision using a simple additive model

$$\hat{x}_t = x_t + \varepsilon_t^x,$$

where $\hat{x}_t$ is the decision that is actually implemented, and ε_t^x captures the difference between what was requested, x_t, versus what was implemented, $\hat{x}_t$. We note that ε_t^x would be modeled as an element of W_t, although in practice it is not always observable.

It is important to distinguish between uncertainty in how a decision (or control) is implemented from other sources of uncertainty because of potential nonlinearities in how the decision affects the results.

10.1.9 Communication errors and biases

In a multiagent system, one agent might communicate location or status to another agent, but this information can contain errors (a drone might not know its exact location) or biases (a fleet driver might report being on the road for fewer hours in order to be allowed to driver longer). In supply chain management, an engine manufacturer may send inflated production targets to suppliers to encourage suppliers to have enough inventory to handle problems, say, in the quality of parts that require more returns.

10.1.10 Algorithmic instability

A more subtle form of uncertainty is one that we refer to as algorithmic uncertainty. We use this category to describe uncertainty that is introduced by the algorithm used to solve a problem, which may also be partly attributable to the model itself. Three examples of how algorithmic uncertainty arises are

- Algorithms that depend on Monte Carlo sampling.
- Algorithms that exhibit sensitivity to small changes in the input data.
- Algorithms that produce different results even when run on exactly the same data, possibly due to variations in run times for a parallel implementation of an algorithm.

A more obvious source of variability that arises in the context of stochastic optimization algorithms are those which depend on Monte Carlo sampling. For example, we are often estimating a parameter, call it μ, which is the mean of a random variable R. We can write this task as an optimization problem

An example of an algorithm that depends on Monte Carlo sampling is a stochastic gradient algorithm. To illustrate, assume that we want to solve the following problem:

$$\min_{\mu} F(\mu) = \mathbb{E}F(\mu, R) = \mathbb{E}\frac{1}{2}(\mu - R)^2.$$

What this problem is doing is finding the mean of a random variable R by finding the value μ that minimizes the expected square of the deviation between μ and R. We could solve this using a gradient procedure such as

$$\mu^{n+1} = \mu^n - \alpha \nabla_\mu F(\mu^n),$$

but imagine that we cannot compute the expectation. As an alternative, we can employ a *stochastic gradient* where we take a Monte Carlo sample of R (which we describe below), and find the gradient of $F(\mu^n, R^{n+1})$, giving us the algorithm

$$\begin{aligned} \mu^{n+1} &= \mu^n + \alpha_n \nabla_\mu F(\mu^n, R^{n+1}) \\ &= \mu^n + \alpha_n(\mu^n - R^{n+1}). \end{aligned}$$

Under some simple conditions, this algorithm is guaranteed to converge to the optimal solution, which happens to be the mean of the random variable R. However, the behavior of the algorithm depends on the sequence of samples $R^1, R^2, \ldots, R^n, \ldots$. If we run this algorithm repeatedly for N iterations, we will get a range of values for μ^N as our estimate of the mean of R. The choice of this particular algorithmic strategy introduces uncertainty in the estimate of the solution.

The second type of algorithmic uncertainty arises due to the sensitivity that many deterministic optimization problems exhibit. Small changes in the input data can produce wide swings in the solution, although often there is little or no change in the objective function. Thus, we may solve an optimization problem (perhaps this might be a linear program) that depends on a parameter θ. Let $F(\theta)$ be the optimal objective function and let $x(\theta)$ be the optimal solution. Small changes in θ can produce large (and unpredictable) changes in $x(\theta)$, which introduces a very real form of uncertainty.

The third type of uncertainty arises primarily with complex problems such as large integer programs that might take advantage of parallel processing. The behavior of these algorithms depends on the performance of the parallel processors, which can be affected by the presence of other jobs on the system. As a result, we can observe variability in the results, even when applied to exactly the same problem with the same data.

Algorithmic uncertainty is in the same class as experimental uncertainty, thus we defer to the discussion there for a description of how to model it.

10.1.11 Goal uncertainty

Many problems involve balancing multiple, competing objectives, such as putting different priorities on cost versus service, profits versus risk. One way to model this is to assume a linear utility function of the form

$$U(S,x) = \sum_{\ell \in \mathcal{L}} \theta_\ell phi_\ell(S,x),$$

where S is our state variable, x is a decision, and $(\phi_\ell(S,x))_{\ell \in \mathcal{L}}$ is a set of features that capture the different metrics we use to evaluate a system. The vector $(\theta_\ell)_{\ell \in \mathcal{L}}$ captures the weight we put on each feature. One way to model goal uncertainty is to represent θ as being uncertainty.

Another form of uncertainty might arise when we do not know all the features $\phi(S,x)$. For example, we may not even be aware that a reason to assign a particular driver to move a customer is that the customer is going to a location near the home of the driver. A human dispatcher might know this through personal interactions with the driver, but a computer might not. The result could then be a disagreement between a computer recommendation and what a human wants to do.

10.1.12 Political/regulatory uncertainty

Uncertainty in political will (which might affect the likelihood that clean energy subsidies will be maintained) and regulatory uncertainties (will there be a change in mileage standards for cars) can have a major impact on long-term investment decisions.

10.1.13 Discussion

Careful readers will notice some overlap between these different types of uncertainty. Observational uncertainty, which refers specifically to errors in the direct observation of a parameter, and inferential uncertainty, which refers to errors in our ability to make inferences about models and parameters indirectly from data, represents one example, but we feel that it is useful to highlight the distinction. Prognostic uncertainty is also a form of inferential uncertainty, but we think it helps to distinguish between uncertainty in the state of the system now (e.g. how the market responds to prices now) and information that we expect to arrive in the future.

10.2 CREATING RANDOM PROCESSES

Somewhere in stochastic optimization we usually end up needing to compute an expectation, as we found in chapter 9 when we formulated our objective function as

$$\min_{\pi} \mathbb{E} \sum_{t=0}^{T} C(S_t, X_t^{\pi}(S_t)).$$

With rare exceptions, we will not be able to compute the expectation, and instead we have to resort to sampling, which can be accomplished in one of rheww ways:

- Monte Carlo sampling - Here we use powerful algorithms that have been developed for generating individual samples from a known distribution. This is the approach used when we are optimizing a simulated problem in the computer.

- Numerical simulations - We may have a (typically large) computer model of a complex process. The simulation may be of a physical system such as a supply chain or an asset allocation model. Some simulation models can require extensive calculations (a single sample realization could take hours or days on a computer). We can use such simulations as a source of observations similar to observations from real-world environments.

- Observational sampling - This is where we use observations from an exogenous process, most commonly referred to as the "real world," to generate sample realizations.

Often, we create simulated versions of the real world in order to test algorithms, with the understanding that the simulated source of observations will be replaced with exogenous observations. It is important to understand whether this is the eventual plan, since some policies depend on having access to an underlying model.

10.2.1 Sample paths

In chapter 9, section 9.7.2, we showed that we could write the value of a policy as

$$F^{\pi} = \mathbb{E}^{\pi} \sum_{t=0}^{T} C(S_t, X_t^{\pi}(S_t)). \tag{10.8}$$

We then wrote this as a simulation using

$$F^{\pi}(\omega) = \sum_{t=0}^{T} C(S_t(\omega), X_t^{\pi}(S_t(\omega))), \tag{10.9}$$

where the states are generated according to $S_{t+1}(\omega) = S^M(S_t(\omega), X_t^{\pi}(S_t(\omega)), W_{t+1}(\omega))$. In this section, we illustrate our notation for representing sample paths more carefully.

We start by assuming that we have constructed 10 potential realizations of price paths $p_t,\ t = 1, 2, \ldots, 8$, which we have shown in table 10.1. Each sample path is a particular set of outcomes of the p_t for all time periods. We index each potential set of outcomes by ω, and let Ω be the set of all sample paths where, for our example, $\Omega = \{1, 2, \ldots, 10\}$. Thus, $p_t(\omega^n)$ would be the price for sample path ω^n at time t. For example, referring to the table we see that $p_2(\omega^4) = 45.67$.

	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$
ω^n	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
ω^1	45.00	45.53	47.07	47.56	47.80	48.43	46.93	46.57
ω^2	45.00	43.15	42.51	40.51	41.50	41.00	39.16	41.11
ω^3	45.00	45.16	45.37	44.30	45.35	47.23	47.35	46.30
ω^4	45.00	45.67	46.18	46.22	45.69	44.24	43.77	43.57
ω^5	45.00	46.32	46.14	46.53	44.84	45.17	44.92	46.09
ω^6	45.00	44.70	43.05	43.77	42.61	44.32	44.16	45.29
ω^7	45.00	43.67	43.14	44.78	43.12	42.36	41.60	40.83
ω^8	45.00	44.98	44.53	45.42	46.43	47.67	47.68	49.03
ω^9	45.00	44.57	45.99	47.38	45.51	46.27	46.02	45.09
ω^{10}	45.00	45.01	46.73	46.08	47.40	49.14	49.03	48.74

Table 10.1 Illustration of a set of sample paths for prices all starting at \$45.00.

One reason that we may generate information on the fly is that it is easier to implement in software. For example, it avoids generating and storing an entire sample path of observations. However, another reason is that random information may depend on the current state, a setting we address next.

10.2.2 State/action dependent processes

Imagine that we are looking to optimize an energy system in the presence of increasing contributions from wind and solar energy. It is reasonable to assume that the available energy from wind or solar, which we represent generically as W_t, is not affected by any decision we make. We could create a series of sample paths of wind, which we could denote by $\hat{\omega} \in \hat{\Omega}$, where each sequence $\hat{\omega}$ is a set of outcomes of $W_1(\hat{\omega}), \ldots, W_T(\hat{\omega})$. These sample paths could be stored in a dataset and used over and over.

There are a number of examples where exogenous information depends on the state of the system. Some examples include:

■ **EXAMPLE 10.1**

Imagine the setting where a patient is being given a cholesterol lowering drug. We have to decide the dosage (10mg, 20mg, . . .), and then we observe blood pressure and whether the patient experiences any heart irregularities. The observations represent the random information, but these observations are influenced by the prior dosage decisions.

■ **EXAMPLE 10.2**

A utility has to decide whether to replace a transformer or to continue monitoring it. As the transformer ages, it is more likely to fail. The random failures depend on the age of the transformer, along with observations of stress events such as voltage surges.

■ **EXAMPLE 10.3**

The price of oil reflects oil inventories. As inventories rise, the market recognizes the present of surplus inventories which depresses prices. Decisions about how much oil to store affects the exogenous changes in market prices.

In some cases, the random information depends on the decision being made at time t. For example, imagine that we are a large investment bank buying and selling stock. Large buy and sell orders will influence the price. Imagine that we place a (large) order to sell x_t shares of stock, which will clear the market at a random price

$$p_{t+1}(x_t) = p_t - \theta x_t + \varepsilon_{t+1},$$

where θ captures the impact of the order on the market price. We are not able to directly observe this effect, so we create a single random variable $\hat{p}_{t+1}$ that captures the entire change in price, given by

$$\hat{p}_{t+1} = -\theta x_t + \varepsilon_{t+1}.$$

Thus, our random variable $\hat{p}_{t+1}$ depends on the decision x_t.

We can model problems where the exogenous information W_{t+1} depends on the action x_t as if it were depending on the post-decision state $S_t^x = (S_t, x_t)$. However, since it is the sales x_t itself that influences the change in price, it is important that x_t be captured explicitly in the post-decision state.

Whether the exogenous information depends on the state or the action, it depends on the policy, since the state at time t reflects prior decisions.

10.3 TYPES OF DISTRIBUTIONS

While it is easy to represent random information as a single variable such as W_t, it is important to realize that random variables can exhibit very different behaviors. The major classes of distributions that we have encountered in our work include:

- Exponential (or geometric) families of random variables, which are the most familiar - These include the continuous distributions such as normal (or Gaussian) distributions, log normal, exponential and gamma distributions, and discrete distributions such as the Poisson distribution, geometric distribution, and the negative binomial distributions. We also include in this class the uniform distribution (continuous or discrete).

- Heavy-tailed distributions - Price processes are a good example of variability that tends to exhibit very high standard deviations. An extreme example is the Cauchy distribution which has infinite variance.

- Spikes - These are infrequent but extreme observations. For example, electricity prices periodically spike from typical prices in the range of 20 to 50 dollars per megawatt, to prices of 300 to 1000 dollars per megawatt for very short intervals (perhaps 5 to 10 minutes).

- Rare events - Rare events are similar to spikes, but are characterized not by extreme values but rather by events that may happen, but happen rarely. For example, failures of jet engines are quite rare, but they happen, requiring that the manufacturer hold spares.

- Bursts - Bursts describe processes such as snow or rain, power outages due to extreme weather, or sales of a product where a new product, advertising or price reduction can produce a rise in sales over a period of time. Bursts are characterized by a sequence of observations over a short period of time.
- Regime shifting - A data series may move from one regime to another as the world changes. Data does not always revert to a mean.
- Hybrid/compound distributions - There are problems where a random variable is drawn from a distribution with a mean which is itself a random variable. The mean of a Poisson distribution, perhaps representing people clicking on an ad, might have a mean which itself is a random variable reflecting the behavior of competing ads.

Below we introduce methods for sampling from these distributions.

10.4 MONTE CARLO SIMULATION

We now address the problem of generating random variables from known probability distributions using a process known as Monte Carlo sampling. Although most software tools come with functions to generate observations from major distributions, it is often necessary to customize tools to handle more general distributions.

There is an entire field that focuses on developing and using tools based on the idea of Monte Carlo simulation, and our discussion should be viewed as little more than a brief introduction.

10.4.1 Generating uniform $[0, 1]$ random variables

Arguably the most powerful tool in the Monte Carlo toolbox is the ability to use the computer to generate random numbers that are uniformly distributed between 0 and 1. This is so important that most computer languages and computing environments have a built-in tool for generating uniform $[0, 1]$ random variables. While we strongly recommend using these tools, it is useful to understand how they work. It starts with a simple recursion that looks like

$$R^{n+1} \leftarrow (a + bR^n) \text{ mod } (m),$$

where a and b are very large numbers, while m might be a number such as $2^{64} - 1$ (for a 64 bit computer), or perhaps $m = 999,999,999$. For example, we might use

$$R^{n+1} \leftarrow (593845395 + 2817593R^n) \quad \text{mod } (999999999).$$

This process simulates randomness because the arithmetic operation $(a + bR)$ creates a number much larger than m, which means we are taking the low order digits, which move in a very random way.

We have to initialize this with some starting variable R^0 called the *random number seed.* If we fix R^0 to some number (say, 123456), then every sequence $R^1, R^2, \ldots$ will be exactly the same (some computers use an internal clock to keep this from happening, but sometimes this is a desirable feature). If a and b are chosen carefully, R^n and R^{n+1} will appear (even under careful statistical testing) to be independent.

Due to the $\mod$ function, all the values of R^n will be between 0 and 999999999. This is convenient because it means if we divide each of them by 999999999, we get a sequence of numbers between 0 and 1. Thus, let

$$U^n = \frac{R^n}{m}.$$

While this process looks easy, we caution readers to use built-in functions for generating random variables, because they will have been carefully designed to produce the required independence properties. Every programming language comes with this function built in. For example, in Excel, the function `Rand()` will generate a random number between 0 and 1 which is both uniformly distributed over this interval, as well as being independent (a critical feature).

Below, we are going to exploit our ability to generate a sequence of uniform $[0, 1]$ random variables to generate a variety of random variables which we denote $W^1, \ldots, W^n, \ldots$. We refer to the sequence W^n as a Monte Carlo sample, while modeling using this sample is referred to as Monte Carlo simulation.

There is a wide range of probability distributions that we may draw on to simulate different types of random phenomena, so we are not even going to attempt to provide a comprehensive list of probability distributions. However, we are going to give a summary of some major classes of distributions, primarily as a way to illustrate different methods for generating random observations.

10.4.2 Uniform and normal random variable

Now that we can generate random numbers between 0 and 1, we can quickly generate random numbers that are uniform between a and b using

$$X = a + (b - a)U.$$

Below we are going to show how we can use our ability to generate (0,1) random variables to generate random variables from many other distributions. However, one important exception is that we cannot easily use this capability to generate random variables that are normally distributed.

For this reason, programming languages also come with the ability to generate random variables Z that are normally distributed with mean 0 and variance 1. With this capability, we can generate random variables that are normally distributed with mean μ and variance σ^2 using the sample transformation

$$X = \mu + \sigma Z.$$

We can take one more step. While we will derive tremendous value from our ability to generate a sequence of *independent* random variables that are uniformly distributed on $[0, 1]$, we often have a need to generate a sequence of *correlated* random variables that are normally distributed. Imagine that we need a vector X

$$X = \begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_N \end{pmatrix}.$$

Now assume that we are given a covariance matrix Σ where $\Sigma_{ij} = Cov(X_i, X_j)$. Just as we use σ above (the square root of the variance σ^2), we are going to take the "square root" of Σ by taking its Cholesky decomposition, which produces an upper right-triangular matrix. In Matlab, this can be done using

$$C = \texttt{chol}(\Sigma).$$

The matrix C satisfies

$$\Sigma = CC^T,$$

which is why it is sometimes viewed as the square root of Σ.

Now assume that we generate a column vector Z of N independent, normally distributed random variables with mean 0 and variance 1. Let μ be a column vector of $\mu_1, \ldots, \mu_N$ which are the means of our vector of random variables. We can now generate a vector of N random variables X with mean μ and covariance matrix Σ using

$$\begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_N \end{pmatrix} = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_N \end{pmatrix} + C \begin{pmatrix} Z_1 \\ Z_2 \\ \vdots \\ Z_N \end{pmatrix}.$$

To illustrate, assume our vector of means is given by

$$\mu = \begin{bmatrix} 10 \\ 3 \\ 7 \end{bmatrix}.$$

Assume our covariance matrix is given by

$$\Sigma = \begin{bmatrix} 9 & 3.31 & 0.1648 \\ 3.31 & 9 & 3.3109 \\ 0.1648 & 3.3109 & 9 \end{bmatrix}.$$

The Cholesky decomposition computed by MATLAB using $C = \texttt{chol}(\Sigma)$ is

$$C = \begin{bmatrix} 3 & 1.1033 & 0.0549 \\ 0 & 3 & 1.1651 \\ 0 & 0 & 3 \end{bmatrix}.$$

Imagine that we generate a vector Z of independent standard normal deviates

$$Z = \begin{bmatrix} 1.1 \\ \text{-0.57} \\ 0.98 \end{bmatrix}.$$

Using this set of sample realizations of Z, a sample realization u would be

$$u = \begin{bmatrix} 10.7249 \\ 2.4318 \\ 9.9400 \end{bmatrix}.$$

10.4.3 Generating random variables from inverse cumulative distributions

Assume we have a distribution with density $f_X(x)$ and cumulative distribution $F_X(x)$, and let $F_X^{-1}(u)$ be the inverse, which means that $x = F_X^{-1}(u)$ is the value of x such that the probability that $X \leq x$ is equal to u (it helps if $0 \leq u \leq 1$). There are some distributions where $F_X^{-1}(u)$ can be found analytically, but computing this numerically can also be quite practical. We now use the following trick from probability. Let U be a random variable that is uniform over the interval $[0,1]$. Then $X = F_X^{-1}(U)$ is a random variable that has the distribution $X \sim f_X(x)$.

A simple example of this result is the case of an exponential density function $\lambda e^{-\lambda x}$ with cumulative distribution function $1 - e^{-\lambda x}$. Setting $U = 1 - e^{-\lambda x}$ and solving for x gives

$$X = -\frac{1}{\lambda} \ln(1 - U).$$

Since $1 - U$ is also uniformly distributed between 0 and 1, we can use

$$X = -\frac{1}{\lambda} \ln(U).$$

We can generate outputs from a gamma distribution given by

$$f(x|k,\theta) = \frac{x^{k-1} e^{-\frac{x}{\theta}}}{\theta^k \Gamma(k)}.$$

$\Gamma(k)$ is the gamma function, with $\Gamma(k) = (k-1)!$ if k is integer. The gamma distribution is created by summing k exponential distributions, each with mean $(k\lambda)^{-1}$. This can be simulated by simply generating k random variables with an exponential distribution and adding them together.

A special case of this result allows us to generate binomial random variables. First sample U which is uniform on [0,1], and compute

$$R = \begin{cases} 1 & \text{if } U < p \\ 0 & \text{otherwise.} \end{cases}$$

R will have a binomial distribution with probability p. The same idea can be used to generate a geometric distribution, which is given by (for $x = 0, 1, \ldots$)

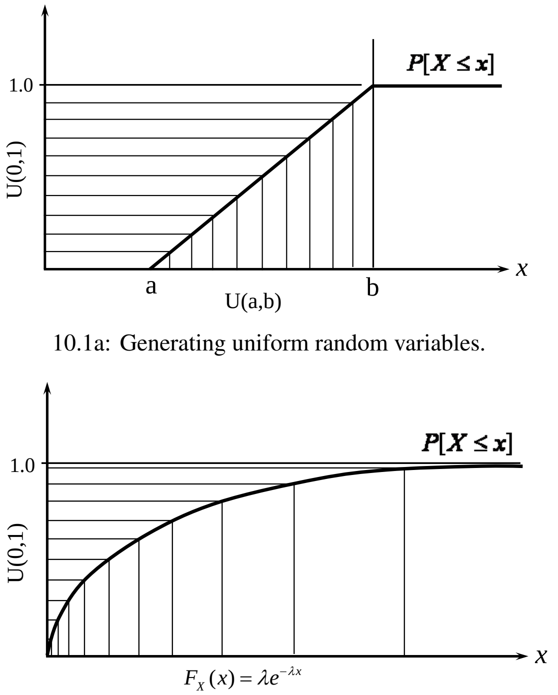
$$\mathbb{P}(X \leq x) = 1 - (1-p)^{k+1}.$$

Now generate U and find the largest k such that $1 - (1-p)^{k+1} \leq U$.

Figure 10.1 illustrates using the inverse cumulative-distribution method to generate both uniformly distributed and exponentially distributed random numbers. After generating a uniformly distributed random number in the interval [0,1] (denoted $U(0,1)$ in the figure), we then map this number from the vertical axis to the horizontal axis. If we want to find a random number that is uniformly distributed between a and b, the cumulative distribution simply stretches (or compresses) the uniform (0,1) distribution over the range (a,b).

10.4.4 Inverse cumulative from quantile distributions

This same idea can be used with a quantile distribution (which is a form of nonparametric distribution). Imagine that we compile our cumulative distribution from data. For example,



10.1a: Generating uniform random variables.

10.1a: Generating exponentially-distributed random variables.

Figure 10.1 Generating uniformly and exponentially distributed random variables using the inverse cumulative distribution method.

we might be interested in a distribution of wind speeds. Imagine that we collect a large sample of observations $X_1, \ldots, X_n, \ldots, X_N$, and further assume that they are sorted so that $X_n \leq X_{n+1}$. We would then let $F_X(x)$ be the percentage of observations that are less than or equal to x. The inverse cumulative is computed by simply associating $f_n = F_X(x_n)$ with each observation x_n. Now, if we choose a uniform random number U, we simply find the smallest value of n such that $f_n \leq U$, and then output X_n as our generated random variable.

10.4.5 Distributions with uncertain parameters

Imagine that we have the problem of optimizing the price charged for an airline or hotel given the random requests from the market. It is reasonable to assume that the arrival process is described by a Poisson arrival process with rate λ customers per day. However, in most settings we do not know λ.

One approach is to assume that λ is described by yet another probability distribution. For example, we might assume that λ follows a gamma-distribution, which is parameterized by (k, θ). Now, instead of having to know λ, we just need to choose (k, θ), which are referred to as *hyperparameters*. Introducing a belief on unknown parameters introduces more parameters for fitting a distribution. For example, if λ is the expected number of arrivals per day, then the variance of the number of arrivals is also λ, but it is quite likely that the variance is much higher. We can tune the hyperparameters (k, θ) so that we still match the mean but produce a variance closer to what we actually observe.

Consider, for example, the problem of sampling Poisson arrivals describing the process of booking rooms for a hotel for a particular date. For simplicity, we are going to assume that the booking rate is a constant λ over the interval $[0, T]$ where T is the date where people would actually stay in the room (in reality, this rate would vary over time). If N_t is the number of customers booking rooms on day t, the probability distribution of N_t would be given by

$$\mathbb{P}[N_t = i] = \frac{\lambda^i e^{-\lambda}}{i!}.$$

We can generate random samples from this distribution using the methods presented earlier.

Now assume that we are uncertain about λ. We might assume that it has a beta distribution which is given by

$$f(x : \alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1},$$

where $\Gamma(k) = (k-1)!$ (if k is integer). The beta distribution takes on a variety of shapes over the domain $0 \leq x \leq 1$ (check out the shapes on Wikipedia). Assume that when we observe bookings, we find that N_t has a mean μ and variance σ^2. If the arrival rate λ were known, we would have $\mu = \sigma^2 = \lambda$. However, in practice we often find that $\sigma^2 > \mu$, in which case we can view λ as a random variable.

To find the mean and variance of λ, we start by observing that

$$\mathbb{E}N_t = \mathbb{E}\{\mathbb{E}\{N_t|\lambda\}\} = \mathbb{E}\lambda = \mu.$$

Finding the variance of λ is a bit harder. We start with the identity

$$\begin{aligned} \mathit{Var}N_t &= \sigma^2 \\ &= \mathbb{E}N_t^2 - (\mathbb{E}N_t)^2. \end{aligned} \tag{10.10}$$

This allows us to write

$$\begin{aligned} \mathbb{E}N_t^2 &= \mathit{Var}N_t + (\mathbb{E}N_t)^2 \\ &= \sigma^2 + \mu^2. \end{aligned}$$

We then use

$$\begin{aligned} \mathbb{E}N_t &= \mathbb{E}\{\mathbb{E}\{N_t|\lambda\}\} \\ &= \mathbb{E}\lambda, \\ &= \mu. \\ \mathbb{E}N_t^2 &= \mathbb{E}\{\mathbb{E}\{N_t^2|\lambda\}\} \\ &= \mathbb{E}\{\lambda + \lambda^2\} \\ &= \mu + (\mathit{Var}\lambda + \mu^2). \end{aligned}$$

We can now write

$$\begin{aligned} \sigma^2 + \mu^2 &= \mu + (\mathit{Var}\lambda + \mu^2), \\ \mathit{Var}\lambda &= \sigma^2 - \mu. \end{aligned}$$

So, given the mean μ and variance σ^2 of N_t, we can find the mean and variance of λ.

The next challenge is to find the parameters α and β of our beta distribution, which has mean and variance

$$\begin{aligned} \mathbb{E}X &= \frac{\alpha}{\alpha+\beta}, \\ \mathit{Var}X &= \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}. \end{aligned}$$

We are going to leave as an exercise to the reader to decide how to pick α and β so that the moments of our beta-distributed random variable X match the moments of λ.

The parameters α and β are called *hyperparameters* as they are distributional parameters that describe the uncertainty in the arrival rate parameter λ. α and β should be chosen so that the mean of the beta distribution closely matches the observed mean μ (which would be the mean of λ). Less critical is matching the variance, but it is important to reasonably replicate the variance σ^2 of N_t.

Once we have fit the beta distribution, we can run simulations by first simulating a value of λ from the beta distribution. Then, given our sampled value of λ (call it $\hat{\lambda}$), we would sample from our Poisson distribution using arrival rate $\hat{\lambda}$.

10.5 SAMPLING VS. SAMPLED MODELS

Monte Carlo sampling is without question the most powerful tool in our toolbox for dealing with uncertainty. In this section, we illustrate three ways of performing Monte Carlo sampling: 1) iterative sampling, 2) solving a static, sampled model, and 3) sequentially solving a sampled model with adaptive learning.

10.5.1 Iterative sampling: A stochastic gradient algorithm

Imagine that we are interested in solving the problem

$$\begin{aligned} F(x) &= \mathbb{E}F(x,W) & (10.11) \\ &= \mathbb{E}\{p\min\{x,\hat{D}\} - cx\}, & (10.12) \end{aligned}$$

where $W = \hat{D}(\omega)$ is a sample realization of the demand $\hat{D}$, drawn from a full set of outcomes Ω. We could search for the best x using a classical stochastic gradient algorithm such as

$$x^{n+1} = x^n - \alpha_n \nabla_x F(x^n, \hat{D}(\omega^{n+1})), \tag{10.13}$$

where

$$\nabla_x F(x,\hat{D}) = \begin{cases} p-c & x > \hat{D}, \\ -c & x \le \hat{D}. \end{cases} \tag{10.14}$$

$\nabla_x F(x,\hat{D})$ is called a *stochastic gradient* because it depends on the random variable $\hat{D}$. Under some conditions (for example, the stepsize α_n needs to go to zero, but not too quickly), we can prove that this algorithm will asymptotically converge to the optimal solution.

10.5.2 Static sampling: Solving a sampled model

A sampled version of this problem, on the other hand, involves picking a sample $\hat{\Omega} = \{\omega^1, \ldots, \omega^N\}$. We then solve

$$\min_\theta \frac{1}{N} \sum_{n=1}^{N} F(\theta|\omega^n). \tag{10.15}$$

This is actually a deterministic problem, although one that is much larger than the original stochastic problem. For many applications, equation (10.15) can be solved using a deterministic solver. The quality of the solution to (10.15) compared to the optimal solution of the original problem (10.11) depends on the application. The stochastic gradient update (10.13) can be much easier to compute than solving the sampled problem (10.15).

In practice, stochastic gradient algorithms require tuning the stepsize sequence α_n which can be quite frustrating. On the other hand, stochastic gradient algorithms can be implemented in an online fashion (e.g. through field observations) while the objective (10.15) is a strictly offline approach. There is a rich theory showing that the optimal solution of (10.15), x^N, asymptotically approaches the true optimal (that is, the solution of the original problem (10.11)) as N goes to infinity, but the algorithm is always applied to a static sample $\hat{\Omega}$. Unlike our stochastic gradient algorithm in the previous section, there is no notion of asymptotic convergence (although in practice we will typically stop our stochastic gradient algorithm after a fixed number of iterations).

10.5.3 Sampled representation with Bayesian updating

We close our discussion with an illustration of using a sampled model where we are uncertain about the parameters of the model. We then run experiments sequentially and update our belief about the probability that each sampled parameter value is correct. Imagine, for example, that we we are solving a stochastic revenue management problem for airlines where we assume that the customers arrive according to a Poisson process with rate λ. The problem is that we are not sure of the arrival rate λ. We assume that the true arrival rate is one of a set of values $\lambda_t^1, \ldots, \lambda_t^K$, where each is true with probability q_t^k. The vector q_t captures our belief about the true parameters, and can be updated using a simple application of Bayes theorem. Let $N(\lambda)$ be a Poisson random variable with mean λ, and let N_{t+1} be the observed number of arrivals between t and $t+1$. We can update q_t using

$$q_{t+1}^k = \frac{\mathbb{P}(N(\lambda) = N_{t+1}|\lambda = \lambda^k) q_t^k}{\sum_{\ell=1}^K q_t^\ell \mathbb{P}(N(\lambda) = N_{t+1}|\lambda = \lambda^\ell)},$$

where

$$\mathbb{P}(N(\lambda) = N_{t+1}|\lambda = \lambda^\ell) = \frac{(\lambda^\ell)^{N_{t+1}} e^{\lambda^\ell}}{N_{t+1}!}.$$

The idea of using a sampled set of parameters is quite powerful, and extends to higher dimensional distributions. However, identifying an appropriate sample of parameters becomes harder as the number of parameters increases.

10.6 CLOSING NOTES

We could have dedicated this entire book to methods for modeling stochastic systems without any reference to decisions or optimization. The study of stochastic systems can be found under names including Monte Carlo simulation and uncertainty quantification, with significant contributions from communities that include statistics, along with the contributions from communities in stochastic optimization that involve the use of Monte Carlo methods, including stochastic search, simulation optimization and stochastic programming.

10.7 BIBLIOGRAPHIC NOTES

Section 10.1 - Some portions of this discussion are based on the Wikipedia entry on uncertainty quantification.

PROBLEMS

10.1 Pick a sequential decision problem of your choosing. Provide a brief explanation, and then list all the types of uncertainty that might arise in this setting (0-5 points will be awarded for each type of uncertainty). It helps if you have some familiarity with the problem class; richer problem settings offer more opportunities for identifying different forms of uncertainty.

CHAPTER 11

POLICIES

Now that we have learned how to simulate an exogenous process $W_1, \ldots, W_t, \ldots$, we return to the challenge of finding a policy that solves our universal objective function from chapter 9

$$\max_{\pi \in \Pi} \mathbb{E}^{\pi} \left\{ \sum_{t=0}^{T} C_t(S_t, X_t^{\pi}(S_t), W_{t+1}) | S_0 \right\}. \tag{11.1}$$

This very elegant formulation begs the obvious question: How in the world do we search over some arbitrary class of policies? This is precisely the reason that this form of the objective function is popular with mathematicians who do not care about computation, or in subcommunities where it is already clear what type of policy is being used. However, equation (11.1) is not widely used, and we believe the reason is that there has not been a natural path to computation. In fact, entire fields have emerged which focus on particular classes of policies.

In this chapter, we address the problem of searching over policies in a general way. Our approach is quite practical in that we organize our search using classes of policies that are widely used either in practice or in the research literature. We start by clarifying one area of confusion, which is the precise meaning of the term "policy" which is popular only in certain subcommunities. A simple definition of a policy is:

Definition 11.0.1. *A* **policy** *is a rule (or function) that determines a decision given the available information in state S_t.*

Stochastic Optimization and Learning. By Warren B. Powell

The problem with the concept of a policy is that it refers to *any* method for determining an action given a state, and as a result it covers a wide range of algorithmic strategies, each suited to different problems with different computational requirements. While we avoided the use of policies in chapter 4 for problems where we could compute the expectation exactly (or a sampled approximation of the expectation), the concept of designing policies pervades any adaptive learning algorithm (whether it is derivative-based of derivative-free), as well as the entire family of problems with state dependent costs and constraints.

For this reason, we return again to the fundamental classes of policies that we have touched on earlier.

11.1 CLASSES OF POLICIES

There are two fundamental strategies for creating policies. These are given by

Policy search - Here we are using equation (11.1) directly to search over a) classes of policies and b) parameters that characterize a particular class.

Policies based on lookahead approximations - These are policies that approximate (sometimes exactly) the value of being in a state in the future, to allow us to understand the impact of a decision made now.

Both of these can lead to optimal policies under certain circumstances, but only in special cases where we can exploit structure. Since these are relatively rare, a variety of approximation strategies have evolved.

Each of these two strategies serve as a foundation for two (meta)classes of policies. The two classes of policies identified using policy search are:

Policy function approximations (PFAs) - These are analytical functions that map a state to a feasible action. PFAs are typically limited to discrete actions, or low-dimensional (and typically continuous) vectors.

Cost function approximations (CFAs) - Here we maximize a parameterized approximation of a cost function, subject to a (possibly parameterized) approximation of the constraints. CFAs might involve solving large linear or integer programs such as scheduling an airline or planning a supply chain. CFAs have the general form

$$X^{CFA}(S_t|\theta) = \arg\max_{x\in\mathcal{X}_t(\theta)} \bar{C}_t(S_t, x|\theta),$$

where $\bar{C}_t(S_t, x|\theta)$ is a parametrically modified cost function, subject to a parametrically modified set of constraints.

The two classes that are based on lookahead approximations are given by:

Value function approximations (VFAs) - These are policies based on an approximation of the value of being in a state. These have the general form

$$X^{VFA}(S_t|\theta) = \arg\max_{x\in\mathcal{X}_t} \big(C(S_t, x) + \overline{V}_t^x(S_t, x|\theta)\big).$$

Direct lookahead policies (DLAs) - This last class of policies maximizes over an approximate model of current and future costs.

Combined, these create four basic classes of policies that encompass every algorithmic strategy that has been proposed for any sequential stochastic optimization problem.

Some observations:

- The first three classes of policies (PFAs, CFAs and VFAs) introduce four different types of functions we might approximate (we first saw these in chapter 3). These include 1) approximating the function we are maximizing $\mathbb{E}F(x, W)$, 2) the policy $X^\pi(S)$, 3) the objective function or constraints, or 4) the downstream value of being in a state $V_t(S_t)$. Function approximation plays an important role in stochastic optimization, and this brings in the disciplines of statistics and machine learning.

  As we saw in chapter 3, there are three broad strategies used for approximating functions:

  Lookup tables - Also referred to as tabular functions, lookup tables mean that we have a discrete value $\overline{V}(S)$ (or action $X^\pi(S)$) for each discrete state S.

  Parametric representations - These are explicit, analytic functions for $\overline{V}(S)$ or $X^\pi(S)$ which generally involve a vector of parameters that we typically represent by θ. Thus, we might write our value function approximation as

  $$\overline{V}(S|\theta^v) = \sum_{f \in \mathcal{F}} \theta^v_f \phi^v_f(S)$$

  or our policy as

  $$X(S|\theta^x) = \sum_{f \in \mathcal{F}} \theta^x_f \phi^x_f(S)$$

  where $\phi_f(S),\ f \in \mathcal{F}$ is a set of features tuned for approximating the value function or the policy. Neural networks are a class of parametric functions (see section 3.9.3) that are popular in the engineering controls community, where they may be used to approximate either the policy or the value function.

  Nonparametric representations - Nonparametric representations offer a more general way of representing functions, but at a price of greater complexity.

- The last three classes of policies (CFAs, VFAs and DLAs) all use an imbedded $\arg\max$ (or $\arg\min$) which means we have to solve a maximization problem as a step in computing the policy. This maximization (or minimization) problem may be fairly trivial (for example, sorting the value of a small set of actions), or quite complex (some applications require solving large integer programs).

- It is possible to get very high quality results from relatively simple policies if we are allowed to tune them (these would fall under policy search). However, this opens the door to using relatively simple lookahead policies (for example, using a deterministic lookahead) which has been modified by tunable parameters for helping to manage uncertainty.

These four classes of policies encompass all the disciplines that we first introduced in chapter 1. We started to hint at the full range of policies in chapter 7 when we addressed derivative-free stochastic optimization. We are going to revisit this set of policies in the context of the much richer classes of problems that we first illustrated in chapter 8. Our goal is to provide a foundation for designing effective policies for the full modeling framework we introduced in chapter 9.

In the remainder of this chapter, we describe these policies in somewhat more depth, but defer to later chapters for complete descriptions. Skimming this chapter is the best way to get a sense of all four classes of policies. We use a simple application to illustrate that each of these four classes may work best on the same problem class, depending on the specific characteristics of the data.

11.2 POLICY FUNCTION APPROXIMATIONS

It is often the case that we have a very good idea of how to make a decision, and we can design a function (which is to say a policy) that returns a decision which captures the structure of the problem. For example:

■ **EXAMPLE 11.1**

A policeman would like to give tickets to maximize the revenue from the citations he writes. Stopping a car requires about 15 minutes to write up the citation, and the fines on violations within 10 miles per hour of the speed limit are fairly small. Violations of 20 miles per hour over the speed limit are significant, but relatively few drivers fall in this range. The policeman can formulate the problem as a dynamic program, but it is clear that the best policy will be to choose a speed, say $\bar{s}$, above which he writes out a citation. The problem is choosing $\bar{s}$.

■ **EXAMPLE 11.2**

A utility wants to maximize the profits earned by storing energy in a battery when prices are lowest during the day, and releasing the energy when prices are highest. There is a fairly regular daily pattern to prices. The optimal policy can be found by solving a dynamic program, but it is fairly apparent that the policy is to charge the battery at one time during the day, and discharge it at another. The problem is identifying these times.

■ **EXAMPLE 11.3**

A trader likes to invest in IPOs, wait a few days and then sell, hoping for a quick bump. She wants to use a rule of waiting d days at which point she sells.

■ **EXAMPLE 11.4**

A drone can be controlled using a series of actuators that govern the force applied in each of three directions to control acceleration, speed and location (in that order). The logic for specifying the force in each direction can be controlled by a neural network.

■ **EXAMPLE 11.5**

We are holding a stock, and would like to sell it when it goes over a price θ.

■ **EXAMPLE 11.6**

In an inventory policy, we will order new product when the inventory S_t falls below θ_1. When this happens, we place an order $a_t = \theta_2 - S_t$, which means we "order up to" θ_2.

■ **EXAMPLE 11.7**

We might choose to set the output x_t from a water reservoir, as a function of the state (the level of the water) S_t of the reservoir, using a linear function of the form $x_t = \theta_0 + \theta_1 S_t$. Or we might desire a nonlinear relationship with the water level, and use a basis function $\phi(S_t)$ to produce a policy $x_t = \theta_0 + \theta_1 \phi(S_t)$.

The most common type of policy function approximation is some sort of parametric model. Imagine a policy that is linear in a set of basis functions $\phi_f(S_t),\ f \in \mathcal{F}$. For example, if S_t is a scalar, we might use $\phi_1(S_t) = S_t$ and $\phi_2(S_t) = S_t^2$. We might also create a constant basis function $\phi_0(S_t) = 1$. Let $\mathcal{F} = \{0, 1, 2\}$ be the set of three basis functions. Assume that we feel that we can write our policy in the form

$$X^\pi(S_t|\theta) = \theta_0\phi_0(S_t) + \theta_1\phi_1(S_t) + \theta_2\phi_2(S_t). \tag{11.2}$$

Here, the index "π" carries the information that the function is linear in a set of basis functions, the set of basis functions, and the parameter vector θ. Policies with this structure are known as *affine policies* because they are linear in the parameter vector.

The art is coming up with the structure of the policy. The science is in choosing θ, which we do by solving the stochastic optimization problem

$$\max_\theta F^\pi(\theta) = \mathbb{E}^\pi \sum_{t=0}^{T} \gamma^t C(S_t, X^\pi(S_t|\theta)). \tag{11.3}$$

Here, we write $\max_\theta$ because we have fixed the class of policies, and we are now searching within a well-defined space. If we were to write $\max_\pi \ldots$, a proper interpretation would be that we would be searching over different functions (e.g. different sets of basis functions), or perhaps even different classes. Note that we will let π be both the class of policy as well as its parameter vector θ, but we still write $F^\pi(\theta)$ explicitly as a function of θ.

The major challenge we face is that we cannot compute $F^\pi(\theta)$ in any compact form, primarily because we cannot compute the expectation. Instead, we have to depend on Monte Carlo samples. Fortunately, there is a field known as stochastic search to help us with this process. We describe these algorithms in more detail in chapter 12.

Parametric policies are popular because of their compact form, but are largely restricted to stationary problems where the policy is not a function of time. Imagine, for example, a situation where the parameter vector in our policy (11.2) is time dependent, giving us a policy of the form

$$X_t^\pi(S_t|\theta) = \sum_{f \in \mathcal{F}} \theta_{tf}\phi_f(S_t). \tag{11.4}$$

Now, our parameter vector is $\theta = (\theta_t)_{t=0}^T$, which is generally dramatically larger than the stationary problem. Solving equation (11.3) for such a large parameter vector (which would easily have hundreds or thousands of dimensions) becomes intractable unless we can compute derivatives of $F^\pi(\theta)$ with respect to θ.

We cover policy function approximations, and how to optimize them, in much greater depth in chapter 12.

11.3 COST FUNCTION APPROXIMATIONS

Cost function approximations represent a class of policy that has been largely overlooked in the academic literature, yet it is widely used in industry. In a nutshell, CFAs involve solving a deterministic optimization problem that has been modified so that it works well over time, under uncertainty.

To illustrate, we might start with a myopic policy of the form

$$X_t^{Myopic}(S_t) = \arg\max_{x \in \mathcal{X}_t} C(S_t, x), \tag{11.5}$$

where $\mathcal{X}_t$ captures the set of constraints. We emphasize that x may be high-dimensional, with a linear cost function such as $C(S_t, x) = c_t x$, subject to a set of linear constraints:

$$\begin{aligned} A_t x_t &= b_t, \\ x_t &\leq u_t, \\ x_t &\geq 0. \end{aligned}$$

This hints at the difference in the type of problems we can consider with CFAs. A sample application might involve assigning resources (people, machines) to jobs (tasks, orders) over time. Let c_{trj} be the cost (or contribution) of assigning resource r to job j at time t, where c_t is the vector of all assignment costs. Also let $x_{trj} = 1$ if we assign resource r to job j at time t, 0 otherwise. Our myopic policy, which assigns resources to jobs to minimize costs now, may perform reasonably well. Now assume that we would like to see if we could make it work a little better.

We can sometimes improve on a myopic policy by solving a problem with a modified objective function.

$$X_t^{CFA}(S_t|\theta) = \arg\max_{x \in \mathcal{X}_t} \Big(C(S_t, x) + \underbrace{\sum_{f \in \mathcal{F}} \theta_f \phi_f(S_t, x)}_{\text{Cost function correction term}} \Big). \tag{11.6}$$

The new term in the objective is called a “cost function correction term.”

In chapter 13, we discuss a wider range of approximation strategies, including modified constraints and hybrid lookahead policies.

11.4 VALUE FUNCTION APPROXIMATIONS

The next class of policy is based on approximating the value of being in a state resulting from an action we take now. The core idea starts with Bellman’s optimality equation (that we first saw in chapter 2 but study in much greater depth in chapter 14), which is written

$$V_t(S_t) = \max_{x \in \mathcal{X}_t} \big(C(S_t, x) + \gamma \mathbb{E}\{V_{t+1}(S_{t+1})|S_t\} \big). \tag{11.7}$$

where $S_{t+1} = S^M(S_t, x, W_{t+1})$. If we use the post-decision state variable S_t^x,

$$V_t(S_t) = \max_{x \in \mathcal{X}_t} \big(C(S_t, x) + V_t^x(S_t^x) \big), \tag{11.8}$$

where $V_t^x(S_t^x)$ is the (optimal) value of being in post-decision state S_t^x at time t. Chapter 14 deals with problems where Bellman's equation (11.7) (or the post-decision form in (11.8)) can be computed exactly. For the vast range of problems where this is not possible, we can try to replace the value function with some sort of statistical approximation we call $\overline{V}_t(S_t)$.

In later chapters (17 - 19), we are going to address the difficult challenge of estimating value function approximations. Given a VFA, we can quickly create a VFA-based policy. If we use the pre-decision value function, we obtain

$$X_t^{VFA-pre}(S_t) = \arg\max_x \left(C(S_t, x) + \gamma \mathbb{E} \left\{ \overline{V}_{t+1}(S_{t+1}) | S_t \right\} \right), \tag{11.9}$$

where $S_{t+1} = S^M(S_t, x, W_{t+1})$. Often the expectation is problematic since in many settings W_{t+1} is multidimensional. We could overcome this using a sampled model, where we would compute

$$X_t^{VFA-pre}(S_t) = \arg\max_x \left(C(S_t, x) + \gamma \frac{1}{|\hat{\Omega}_{t+1}|} \sum_{\omega \in \hat{\Omega}_t} \overline{V}_{t+1}(S_{t+1}(\omega))) \right), \tag{11.10}$$

where $S_{t+1}(\omega) = S^M(S_t, x, W_{t+1}(\omega))$. If there is a natural (and ideally more compact) post-decision state, then

$$X_t^{VFA-post}(S_t) = \arg\max_x \left(C(S_t, x) + \gamma \overline{V}_t(S_t^x) \right), \tag{11.11}$$

where the post-decision state $S_t^x = S^{M,x}(S_t, x)$ is a deterministic function of the pre-decision state S_t and decision x_t. Clearly, if we can take advantage of a post-decision state, then the post-decision version of a VFA policy in (11.11) is the easiest to use.

Dynamic programming is most frequently illustrated for discrete actions, where we might use action $a \in \mathcal{A}$ instead of x. Discrete actions simplify the process of searching for the best decision and allow us to use lookup table representations of the value function. However, we illustrated a number of applications in chapter 8 involving the management of resources where x might be a high-dimensional vector, as might arise when assigning resources to tasks. In this case, we need to be able to represent the value function in a way that allows the optimization problem to be solved using powerful algorithms from math programming. These typically require that our contribution and value functions be concave (convex if minimizing). We revisit this important problem class in chapter 19. It is in this setting where the post-decision value function is particularly useful since it eliminates the imbedded expectation.

A closely related policy, developed under the umbrella of reinforcement learning within computer science, is to use Q-factors which approximate the value of being in a state S_t and taking discrete action a_t (the strategy only works for discrete actions). Let $\bar{Q}^n(s, a)$ be our approximate value of being in state s and taking action a after n iterations. Q-learning uses some rule to choose a state s^n and action a^n, and then uses some process to simulate a subsequent downstream state s' (which might be observed from a physical system). It then proceeds by computing

$$\hat{q}^n(s^n, a^n) = C(s^n, a^n) + \max_{a'} \bar{Q}^{n-1}(s', a'), \tag{11.12}$$

$$\bar{Q}^n(s^n, a^n) = (1 - \alpha)\bar{Q}^{n-1}(s^n, a^n) + \alpha \hat{q}^n(s^n, a^n). \tag{11.13}$$

Given a set of Q-factors $\bar{Q}^n(s,a)$, the policy is given by

$$A^Q(S_t) = \arg\max_a \bar{Q}^n(S_t,a). \tag{11.14}$$

The appeal of Q-learning based policies is that they do not require knowledge of the transition function (known as the "model" in this community) when finding the best action (as in equation (11.14)). Thus, Q-learning is known as a "model-free" policy (the same is true of our PFA and CFA policies). Q-learning is typically used on problems with relatively small sets of states and actions. If we had access to the one-step transition matrix $p(s'|s,a)$ (which the reinforcement learning community refers to as "the model"), then we would be able to solve these problems exactly using the tools of discrete Markov decision processes, which we introduce in depth in chapter 14.

11.5 DIRECT LOOKAHEAD POLICIES

We save direct lookahead policies for last because this is the most brute-force approach among the four classes of policies. This is the only policy that does not use machine learning to approximate some function. Instead, we resort to *model approximation.*

11.5.1 The basic idea

Imagine that we are in a state S_t. We would like to choose an action x_t that maximizes the contribution $C(S_t,x_t)$ now, plus the value of the state that our action takes us to. Given S_t and x_t, we will generally experience some randomness W_{t+1} that then takes us to state S_{t+1}. The value of being in state S_{t+1} is given by

$$\begin{aligned} V_{t+1}(S_{t+1}) &= \max_\pi \mathbb{E}\left\{\sum_{t'=t+1}^{T} C(S_{t'},X^\pi_{t'}(S_{t'}))|S_t,x_t\right\}. \\ &= \mathbb{E}\left\{\sum_{t'=t+1}^{T} C(S_{t'},X^*_{t'}(S_{t'}))|S_t,x_t\right\}. \end{aligned} \tag{11.15}$$

We could write our optimal policy just as we did above in equation (11.7)

$$X^*(S_t) = \arg\max_{x_t}\big(C(S_t,x_t) + \mathbb{E}\{V_{t+1}(S_{t+1})|S_t,x_t\}\big),$$

but now we are going to recognize that we generally cannot compute the value function $V_{t+1}(S_{t+1})$. Rather than try to approximate this function, we are going to substitute in the definition of $V_{t+1}(S_{t+1})$ from (11.15), which gives us

$$X^*_t(S_t) = \arg\max_{x_t}\left(C(S_t,x_t) + \mathbb{E}\left\{\mathbb{E}\left\{\sum_{t'=t+1}^{T} C(S_{t'},X^*_{t'}(S_{t'}))\middle| S_{t+1}\right\}|S_t,x_t\right\}\right). \tag{11.16}$$

Another way of writing (11.16) is to explicitly imbed the search for the optimal policy in the lookahead portion, giving us

$$X^*_t(S_t) = \arg\max_{x_t}\left(C(S_t,x_t) + \mathbb{E}\left\{\max_\pi \mathbb{E}\left\{\sum_{t'=t+1}^{T} C(S_{t'},X^\pi_{t'}(S_{t'}))\middle| S_{t+1}\right\}|S_t,x_t\right\}\right). \tag{11.17}$$

Equation (11.17) can look particularly daunting, until we realize that this is exactly what we are doing when we solve a decision tree (exercise 11.5 provides a numerical example).

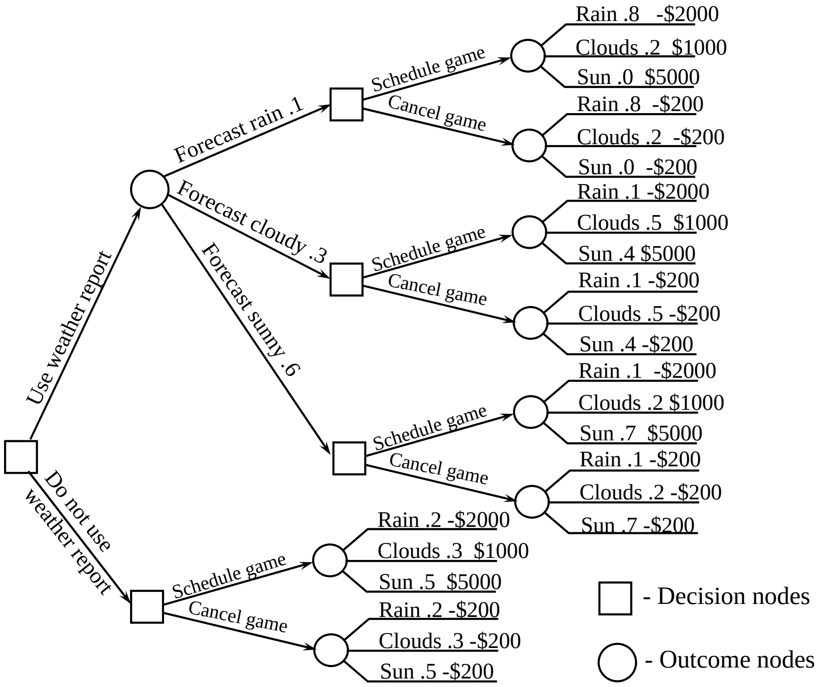
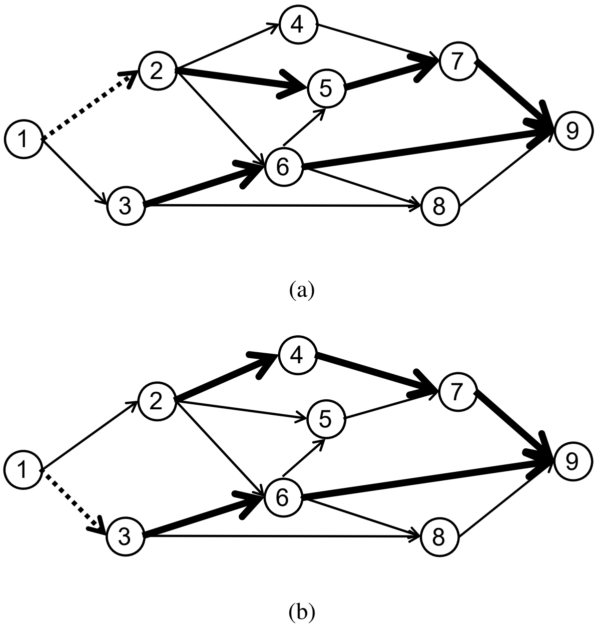


Figure 11.1 Decision tree showing decision nodes and outcome nodes.

We return to our decision tree that we first saw in figure 20.2, which we repeat here for convenience in figure 11.1. Remember that a "decision node" in a decision tree (the squares) corresponds to the state S_t (if we are referring to the first node), or the states $S_{t'}$ for the later nodes. We could use some generic rule $X^{\pi}_{t'}(S_{t'})$ for making a decision, or we can solve the decision tree by stepping backward through the tree to find the optimal action $x^*_{t'}$ for each discrete state $S_{t'}$, which is a lookup table representation for the optimal policy $X^*_{t'}(S_{t'})$. We just have to recognize that $X^{\pi}_{t'}(S_{t'})$ refers to *some* rule for choosing an action out of node $S_{t'}$, while $X^*_{t'}(S_{t'})$ is the best action out of node $S_{t'}$.

To parse equation (11.17), the first expectation, which is conditioned on state S_t and action x_t, is over the first set of random outcomes out of the circle nodes. The inner $\max_{\pi}$ refers generally to the process of finding the best action out of *each* of the remaining decision nodes, before knowing the downstream random outcomes. We then evaluate this policy by taking the expectation over all outcomes.

Another way to help understand equation (11.16) (or (11.17)) is to think about a deterministic shortest path problem. Consider the networks shown in figure 11.2. If we know that we would use the path 2-5-7-9 to get from 2 to 9, we would choose to go from 1 to 2 to take advantage of this path. But if we elect to use a different path out of node 2 (a costlier path), then our decision from node 1 might be to go to node 3.



(a)

(b)

Figure 11.2 (a) Decision to go (1,2) given the path 2-5-7-9. (b) Decision to go (1,3) when path out of node 2 changes.

11.5.2 Modeling the lookahead problem

This hints at one of the most popular ways of approximating the future for a stochastic problem, which is simply to use a deterministic approximation of the future. We can create what we are going call a *deterministic lookahead model*, where we act as if we are optimizing in the future, but only for an approximate model. So we do not confuse the lookahead model with the model we are trying to solve, we are going to introduce two notational devices. First, we are going to use tilde's for state and decision variables, and we are going to index them by t and t', where t refers to the time at which we are making a decision, and t' indexes time within our lookahead model. Thus, a deterministic lookahead model over a horizon $t, \ldots, t+H$, would be formulated as

$$X_t^{LA-Det}(S_t|\theta) = \underset{x_t,(\tilde{x}_{t,t+1},\ldots,\tilde{x}_{t,t+H})}{\arg\max} \left(C(S_t, x_t) + \sum_{t'=t+1}^{t+H} C(\tilde{S}_{tt'}, \tilde{x}_{tt'}) \right). \quad (11.18)$$

Here, we have replaced the model of the problem from time $t+1$ to the end of horizon T with a deterministic approximation that goes out to some truncated horizon $t+H$.

There are special cases where we can solve a stochastic lookahead model. One is problems with small numbers of discrete actions, and relatively simple forms of uncertainty. In this case, we can represent our problem using a decision tree such as the one we illustrated in figure 11.1. A decision tree allows us to find the best decision for each node (that is, each state), which is a form of lookup table policy. The problem is that decision trees explode in size for most problems, limiting their usefulness. In chapter 20, we describe methods for formulating and solving stochastic lookahead models using Monte Carlo methods.

While this is the simplest type of lookahead policy, it illustrates the basic idea. We cannot solve the true problem in (11.17), so we introduced a variety of approximations. Deterministic lookahead models tend to be relatively easy to solve (but not always). However, using a deterministic approximation of the future means that we may make decisions now that do not properly prepare us for random events that may happen in the future. Thus, there is considerable interest in solving a lookahead model that recognizes that the future is uncertain.

When dealing with uncertainty, we have to deal with policies we use in the future, rather than decisions. Equation (11.17) says that to find the optimal policy now, we have to know the optimal policies $X^*_{t'}(S_{t'})$ for future time periods. Optimal policies are rarely available in practice, so we are limited to working within a class of suboptimal policies (which could be any of our four classes of policies). For example, imagine that we want to write a direct lookahead policy that we are going to call $X^{DLA}_t(S_t)$ where we use the structure of the policy in (11.17), but where we replace the optimal policy with some simple parametric policy in the future, such as

$$\tilde{X}^{Lin}_t(\tilde{S}_{tt'}|\theta_t) = \theta_{t0} + \theta_{t1}\phi_1(\tilde{S}_{tt'}) + \theta_{t2}\phi_2(\tilde{S}_{tt'}),$$

where $\tilde{S}_{tt'}$ refers to the projected state at time t' when we are making a decision at time t. We are not adopting this as our policy; we are only using this policy to approximate decisions in the future to help us make a good decision now. Of course, we still want the best possible policy as we peek into the future, so this means optimizing the parameter vector θ_t.

Now imagine that we use $\tilde{X}_t(\tilde{S}_{tt'}|\theta_t)$ for the policy we use for peeking into the future. We would, of course, need to search for the best value of θ in the future to help us make the best decision now (at time t), so we index it by time, giving us θ_t, but we assume we are using one θ over the whole horizon $t' = t+1, \ldots, T$ as we peek into the future. Using this type of policy means that our original (and uncomputable) optimal policy in equation (11.16) now becomes

$$X^{LA-Stoch}_t(S_t) = \arg\max_{x_t} \Bigg(C(S_t, x_t) + \tilde{E}\left\{ \max_{\tilde{\theta}_t} \tilde{E}\left\{ \sum_{t'=t+1}^{T} C(\tilde{S}_{tt'}, \tilde{X}^{Lin}_t(\tilde{S}_{tt'}|\tilde{\theta}_t))|\tilde{S}_{t,t+1} \right\} |S_t, x_t \right\} \Bigg). \quad (11.19)$$

We have written the policy $X^{LA-Stoch}_t(S_t)$ assuming that we actually are going to explicitly optimize the policy in the lookahead model as we step forward in time. This optimization, over the lookahead parameter $\tilde{\theta}_t$, would produce an optimal parameter $\tilde{\theta}^*_t(S_t)$ that depends on both time t and on the state S_t we are in when we need to find the best policy.

In practice, of course, no-one would do this. Instead, we would fix $\tilde{\theta}_t(S_t) = \theta$ which then becomes part of the policy $X^{LA-Stoch}_t(S_t|\theta)$, which we would write as

$$X^{LA-Stoch}_t(S_t|\theta) = \arg\max_{x_t} \Bigg(C(S_t, x_t) + \tilde{E}\left\{ \tilde{E}\left\{ \sum_{t'=t+1}^{T} C(\tilde{S}_{tt'}, \tilde{X}^{Lin}_t(\tilde{S}_{tt'}|\theta))|\tilde{S}_{t,t+1} \right\} |S_t, x_t \right\} \Bigg). \quad (11.20)$$

Now we face the problem of tuning θ in $X_t^{LA-Stoch}(S_t|\theta)$ just as we would any parameterized policy.

11.6 HYBRID STRATEGIES

Now that we have identified the four major (meta)classes of policies, we need to recognize that we can also create hybrids by mixing the different classes.

The set of (possibly tunable) myopic policies, lookahead policies, policies based on value function approximations, and policy function approximations represents the core tools in the arsenal for finding effective policies for sequential decision problems. Given the richness of applications, it perhaps should not be surprising that we often turn to mixtures of these strategies.

Cost function approximation with policy function approximations

A major strength of a deterministic lookahead policy is that we can use powerful math programming solvers to solve high-dimensional deterministic models. A challenge is handling uncertainty in this framework. Policy function approximations, on the other hand, are best suited for relatively simple decisions, and are able to handle uncertainty by capturing structural properties (when they can be clearly identified). PFAs can be integrated into high-dimensional models as nonlinear penalty terms acting on individual (scalar) variables.

As an example, consider the problem of assigning resources (imagine we are managing blood supplies) to tasks, where each resource is described by an attribute vector a (the blood type and age) while each task is described by an attribute vector b (the blood type of a patient, along with other attributes such as whether the patient is an infant or has immune disorders). Let c_{ab} be the contribution we assign if we assign a resource of type a to a patient with blood type b. Let R_{ta} be the number of units of blood type a available at time t, and let D_{tb} be the demand for blood b. Finally let x_{tab} be the number of resources of type a assigned to a task of type b. A myopic policy (a form of cost function approximation) would be to solve

$$X^{CFA}(S_t) = \arg\max_{x_t} \sum_{a\in\mathcal{A}} \sum_{b\in\mathcal{B}} c_{ab} x_{tab}.$$

subject to

$$\sum_{b\in\mathcal{B}} x_{tab} \leq R_{ta}, \tag{11.21}$$

$$\sum_{a\in\mathcal{A}} x_{tab} \leq D_{tb}, \tag{11.22}$$

$$x_{tab} \geq 0. \tag{11.23}$$

This policy would maximize the total contribution for all blood assignments, but might ignore issues such as a doctor's preference to avoid using blood that is not a perfect match for infants or patients with certain immune disorders. A doctor's preferences might be expressed through a set of patterns ρ_{ab} which gives the fraction of demand of type b to be satisfied with blood of type a, where $\sum_a \rho_{ab} = 1$. The vector $\rho_{\cdot b} = (\rho_{ab})_{a\in\mathcal{A}}$ can be viewed as a probabilistic policy describing how to satisfy a demand for a unit of blood of type b (it is a form of PFA). In fact, we could replace our CFA policy above and just use this probabilistic policy, which might be based on historical patterns of how patients were served. A strength of our statistically-based PFA is that we could consider a number of patient and blood attributes, and look for patterns that capture the insights of doctors.

This PFA is a low-dimensional rule that expresses a preference for serving a patient of type b with blood type a. Each element ρ_{ab} can be viewed as a low-dimensional function that expresses a preference about a single (a, b) combination, which would not be able to perform tradeoffs between patients. We can combine our myopic cost model (a form of CFA) with our pattern ρ (a PFA) to create a hybrid that would be written

$$X^{CFA-PFA}(S_t|\theta) = \arg\max_{x_t} \sum_{a\in\mathcal{A}} \sum_{b\in\mathcal{B}} \left(c_{ab}x_{tab} + \theta(x_{tab} - D_{tb}\rho_{ab})^2\right),$$

where θ is a tunable parameter that controls the weight placed on the PFA. This can now be optimized using policy search methods.

Lookahead policies with value function approximations

Deterministic rolling horizon procedures offer the advantage that we can solve them optimally, and if we have vector-valued decisions, we can use commercial solvers. Limitations of this approach are a) they require that we use a deterministic view of the future and b) they can be computationally expensive to solve (pushing us to use shorter horizons). By contrast, a major limitation of value function approximations is that we may not be able to capture the complex interactions that are taking place within our optimization of the future.

An obvious strategy is to combine the two approaches. For low-dimensional action spaces, we can use tree search or a roll-out heuristics for H periods, and then use a value function approximation. If we are using a rolling horizon procedure for vector-valued decisions, we might solve

$$X^\pi(S_t) = \arg\max_{x_t,\ldots,x_{t+H}} \sum_{t'=t}^{t+H-1} C(S_{t'}, x_{t'}) + \gamma^H \overline{V}_{t+H}(S_{t+H}),$$

where S_{t+H} is determined by X_{t+H}. In this setting, $\overline{V}_{t+H}(S_{t+H})$ would have to be some convenient analytical form (linear, piecewise linear, nonlinear) in order to be used in an appropriate solver.

The hybrid strategy makes it possible to capture the future in a very precise way for a few time periods, while minimizing truncation errors by terminating the tree with an approximate value function. This is a popular strategy in computerized chess games, where a decision tree captures all the complex interactions for a few moves into the future. Then, a simple point system capturing the pieces lost is used to reduce the effect of a finite horizon.

We note that recent breakthroughs in the use of computers to solve chess or the Chinese game of Go used a hybrid strategy that mixes lookahead policies (using tree search methods we describe in chapter 20), PFAs (basically rules of how to behave based on patterns derived from looking at past games), and VFAs.

Lookahead policies with cost function approximations

A rolling horizon procedure using a deterministic forecast is, of course, vulnerable to the use of a point forecast of the future. For example, we might be planning inventories for our supply chain for iPhones, but a point forecast might allow inventories to drop to zero if this still allows us to satisfy our forecasts of demand. This strategy would leave the supply chain vulnerable if demands are higher than expected, or if there are delivery delays.

This limitation will not be solved by introducing value function approximations at the end of the horizon. It is possible, however, to perturb the forecasts of demands to account for uncertainty. For example, we could inflate the forecasts of demand to encourage holding inventory. We could multiply the forecast of demand $f^D_{tt'}$ at time t' made at time t by a factor $\theta^D_{t'-t}$. This gives us a vector of tunable parameters $\theta^D_1, \ldots, \theta^D_H$ over a planning horizon of length H. Now we just need to tune this parameter vector to achieve good results over many sample paths.

Tree search with rollout heuristic and a lookup table policy

A surprisingly powerful heuristic algorithm that has received considerable success in the context of designing computer algorithms to play games uses a limited tree search, which is then augmented by a rollout heuristic assisted by a user-defined lookup table policy. For example, a computer might evaluate all the options for a chess game for the next four moves, at which point the tree grows explosively. After four moves, the algorithm might resort to a rollout heuristic, assisted by rules derived from thousands of chess games. These rules are encapsulated in an aggregated form of lookup table policy that guides the search for a number of additional moves into the future.

Value function approximation with lookup table or policy function approximation

Assume we are given a policy $\bar{X}(S_t)$, which might be in the form of a lookup table or a parameterized policy function approximation. This policy might reflect the experience of a domain expert, or it might be derived from a large database of past decisions. For example, we might have access to the decisions of people playing online poker, or it might be the historical patterns of a company. We can think of $\bar{X}(S_t)$ as the decision of the domain expert or the decision made in the field. If the action is continuous, we could incorporate it into our decision function using

$$X^\pi(S_t) = \arg\max_x \left(C(S_t, x) + \overline{V}(S^{M,x}(S_t, x)) - \beta(\bar{X}(S_t) - x)^2 \right).$$

The term $\beta(\bar{X}(S_t) - x)^2$ can be viewed as a penalty for choosing actions that deviate from the external domain expert. β controls how important this term is. We note that this penalty term can be set up to handle decisions at some level of aggregation.

Fitting value functions based on lookahead and policy search

Assume we are using a value function approximation $\overline{V}_{t+1}(S_{t+1})$ around the pre-decision state S_{t+1} to create a VFA-based policy

$$X_t^{VFA-hybrid}(S_t) = \arg\max_x \left(C(S_t, x) + \gamma \mathbb{E}\left\{ \overline{V}_{t+1}(S_{t+1}) | S_t \right\} \right).$$

Now assume we are using a parameterized approximation for the value function approximation such as

$$\overline{V}_t(S_t|\theta) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(S_t), \tag{11.24}$$

where $(\phi_f(S_t))_{f\in\mathcal{F}}$ is a user-defined set of features and θ is a set of parameters. Chapters 16-18 cover strategies for approximating value functions in much greater depth under the umbrella of approximate dynamic programming (ADP). These methods can produce good solutions, but classical ADP techniques are hardly perfect, especially when using parameterized approximations such as the linear model in equation (11.24).

An interesting option is to use ADP-based techniques to find an estimate of θ so that $\overline{V}_t(S_t|\theta)$ reasonably approximates the value of being in state S_t. Let this best estimate be designed θ^0, which we are then going to use as the initial estimate in a policy search algorithm that solves

$$\max_\theta F(\theta) = \mathbb{E}\sum_{t=0}^{T} C(S_t, X^{VFA-hybrid}(S_t|\theta)). \tag{11.25}$$

This will typically require the use of one of our derivative-free algorithms that we introduced in chapter 7. Each evaluation of $F(\theta)$ requires running a simulation to obtain a sampled estimate of the objective function on the right hand side of equation (11.25).

We close by noting that when we use ADP algorithms, it is not too hard to compute a time-dependent version of our parametric approximation of the value function, which would be written

$$\overline{V}_t(S_t|\theta_t) = \sum_{f\in\mathcal{F}} \theta_{tf}\phi_f(S_t).$$

If our linear model has 10 parameters and we are optimizing a problem over 50 time periods, this means we have to estimate 500 parameters. Using the techniques of approximate dynamic programming (given in chapters 16-18) this is not a problem. However, performing policy search using the objective function in (11.25) using derivative-free stochastic optimization would be a daunting task.

11.7 RANDOMIZED POLICIES

There are several situations where it is useful to randomize a policy.

Exploration-exploitation - This is easily the most common use of randomized policies. Three popular examples of exploration-exploitation policies are:

Epsilon-greedy exploration This is a popular policy for balancing exploration and exploitation, and can be used for any problem with discrete actions, where the policy has an imbedded $\arg\max_a$ to choose the best discrete action within a set $\mathcal{A}$ (we use action a because this approach does not make sense if the action is continuous or a vector). Let $C(s,a)$ be the contribution from being in state s and taking action a, which might include a value function or a lookahead model. The epsilon-greedy policy chooses an action $a \in \mathcal{A}$ at random with probability ϵ, and chooses the action $\arg\max_{a\in\mathcal{A}} C(s,a)$ with probability $1-\epsilon$.

Boltzmann exploration Let $\bar{Q}^n(s,a)$ be the current estimate of the value of being in state s and taking action a. Now compute the probability of choosing action a according to the Boltzmann distribution

$$P(a|s,\theta) = \frac{e^{\theta\bar{Q}^n(s,a)}}{\sum_{a'\in\mathcal{A}} e^{\theta\bar{Q}^n(s,a')}}.$$

The parameter θ is a tunable parameter, where $\theta = 0$ produces a pure exploration policy, while as θ increases, the policy becomes greedy (choosing the action that appears to be best), which is a pure exploitation policy. The Boltzmann chooses what appears to be the best action with the highest probability, but any action may be chosen. This is the reason it is often called a *soft max* operator.

Excitation Assume that the control x is continuous (and possibly vector-valued). Let Z be a similarly-dimensioned vector of normally distributed random variables with mean 0 and variance 1. An excitation policy perturbs the policy $X^\pi(S_t)$ by adding a noise term such as

$$x_t = X^\pi(S_t) + \sigma Z,$$

where σ is an assumed level of noise.

Thompson sampling As we saw in chapter 7, Thompson sampling uses a prior on the value of $\mu_x = \mathbb{E}F(x, W)$ is $\mu_x \sim N(\bar{\mu}_x^n, \sigma_x^{2,n})$. Now draw $\hat{\mu}_x^n$ from the distribution $N(\bar{\mu}_x^n, \sigma_x^{2,n})$ for each x, and then choose

$$X^{TS}(S^n) = \arg\max_x \hat{\mu}_x^n.$$

Modeling unpredictable behavior - We may be trying to model the behavior of a system with human input. The policy $X^\pi(S_t)$ may reflect perfectly rational behavior, but a human may behave erratically.

Disguising the state In a multiagent system, a decision can reveal private information. Randomization can help to disguise private information.

It is possible to convert any randomized policy into a deterministic one by including a uniformly-distributed random variable U_{t-1} (or the normally-distributed variable Z) to the exogenous information process W_{t-1} so that it becomes a part of the state variable S_t. This random variable can then be used to provide the additional information to make $X^\pi(S_t)$ a deterministic function of the (now expanded) state S_t. However, it is standard to refer to the policies above as "random."

11.8 ILLUSTRATION: AN ENERGY STORAGE MODEL REVISITED

In section 9.8, we presented a model of an energy storage problem. We are going to return to this problem and demonstrate all four classes of policies, along with a hybrid. We are going to further show that *each* of these policies may work best depending on the data. We recommend reviewing the model since we are going to use the same notation.

11.8.1 Policy function approximation

Our policy function approximation is given by

$$X_t^{PFA}(S_t|\theta) = \left\{ \begin{array}{lcl} x_t^{EL} & = & \min\{L_t, E_t\}, \\ x_t^{BL} & = & \left\{ \begin{array}{ll} h_t & \text{If } p_t > \theta^U \\ 0 & \text{If } p_t < \theta^U \end{array} \right. \\ x_t^{GL} & = & L_t - x_t^{EL} - x_t^{BL}, \\ x_t^{EB} & = & \min\{E_t - x_t^{EL}, \rho^{chrg}\}, \\ x_t^{GB} & = & \left\{ \begin{array}{ll} \rho^{chrg} - x_t^{EB} & \text{If } p_t < \theta^L \\ 0 & \text{If } p_t > \theta^L \end{array} \right. \end{array} \right.$$

where $h_t = \min\{L_t - x_t^{EL}, \min\{R_t, \rho^{chrg}\}\}$. This policy is parameterized by (θ^L, θ^U) which determine the price points at which we charge or discharge.

11.8.2 Cost function approximation

The cost function approximation minimizes a one-period cost plus a tunable error correction term:

$$X^{CFA-EC}(S_t|\theta) = \arg\min_{x_t \in \mathcal{X}_t} \big(C(S_t, x_t) + \theta(x_t^{GB} + x_t^{EB} + x_t^{BL})\big), \qquad (11.26)$$

where $\mathcal{X}_t$ is defined by (9.21)-(9.25). We use a linear correction term for simplicity which is parameterized by the scalar θ.

11.8.3 Value function approximation

Our VFA policy uses an approximate value function approximation, which we write as

$$X^{VFA}(S_t) \quad = \quad \arg\min_{x_t \in \mathcal{X}_t} \big(C(S_t, x_t) + \overline{V}_t^x(R_t^x)\big), \qquad (11.27)$$

where $\overline{V}_t^x(R_t^x)$ is a piecewise linear function approximating the marginal value of the post-decision resource state. We use methods described in chapter 19 to compute the value function approximation which exploits the natural convexity of the problem. For now, we simply note that the approximation is quite good.

11.8.4 Deterministic lookahead

The next policy is a deterministic lookahead over a horizon H which has access to a forecast of wind energy.

$$X_t^{LA-DET}(S_t|H) \; = \; \arg\min_{(x_t, \tilde{x}_{t+1,t}, \ldots, \tilde{x}_{t,t+H})} \left(C(S_t, x_t) + \sum_{t'=t+1}^{t+H} C(\tilde{S}_{tt'}, \tilde{x}_{tt'}) \right) (11.28)$$

subject to, for $t' = t, \ldots, T$:

$$\begin{aligned} \tilde{x}_{tt'}^{EL} + \tilde{x}_{tt'}^{EB} &\leq f_{tt'}^{E}, & (11.29)\\ f_{tt'}^{\eta}(\tilde{x}_{tt'}^{GL} + \tilde{x}_{tt'}^{EL} + \tilde{x}_{tt'}^{BL}) &= f_{tt'}^{L}, & (11.30)\\ \tilde{x}_{tt'}^{BL} &\leq \tilde{R}_{tt'}, & (11.31)\\ \tilde{R}_{t,t'+1} - (\tilde{R}_{tt'} + f_{t,t'+1}^{\eta}(\tilde{x}_{tt'}^{GB} + \tilde{x}_{tt'}^{EB}) - \tilde{x}_{tt'}^{BL}) &= f_{t,t'+1}^{R}, & (11.32)\\ \tilde{x}_{tt'} &\geq 0. & (11.33) \end{aligned}$$

We use tilde's on variables in our lookahead model so they are not confused with the same variable in the base model. The variables are also indexed by t, which is when the lookahead model is formed, and t', which is the time period within the lookahead horizon.

11.8.5 Hybrid lookahead-cost function approximation

Our last policy, $X_t^{LA-CFA}(S_t|\theta^L, \theta^U)$, is a hybrid lookahead with a form of cost function approximation in the form of two additional constraints for $t' = t+1, \ldots, T$:

$$\begin{aligned} \tilde{R}_{tt'} &\geq \theta^L, & (11.34)\\ \tilde{R}_{tt'} &\leq \theta^U. & (11.35) \end{aligned}$$

These constraints provide buffers to ensure that that we do not plan on the energy level getting too close to the lower or upper limits, allowing us to anticipate that there will be times when the energy from a renewable source is lower, or higher, than we planned. We note that a CFA-lookahead policy is actually a hybrid policy, combining a deterministic lookahead with a cost function approximation (where the approximation is in the modification of the constraints).

11.8.6 Experimental testing

To test our policies, we created five problem variations:

A) A stationary problem with heavy-tailed prices, relatively low noise, moderately accurate forecasts and a reasonably fast storage device.

B) A time-dependent problem with daily load patterns, no seasonalities in energy and price, relatively low noise, less accurate forecasts and a very fast storage device.

C) A time-dependent problem with daily load, energy and price patterns, relatively high noise, less accurate forecasts using time series (errors grow with the horizon) and a reasonably fast storage device.

D) A time-dependent problem with daily load, energy and price patterns, relatively low noise, very accurate forecasts and a reasonably fast storage device.

E) Same as (C), but the forecast errors are stationary over the planning horizon.

Each problem variation was designed specifically to take advantage of the characteristics of each of our five policies. We tested all five policies on all five problems. In each case, we evaluated the policy by solving the problem using perfect information (this is known as a posterior bound), and then evaluating the policy as a fraction of this posterior bound. The results are shown in table 11.1, where the bold entries (in the diagonal) indicates the policy that worked best on that problem class.

Problem:	PFA	CFA-EC	VFA	LA-DET	LA-CFA
A	**0.959**	0.839	0.936	0.887	0.887
B	0.714	**0.752**	0.712	0.746	0.746
C	0.865	0.590	**0.914**	0.886	0.886
D	0.962	0.749	0.971	**0.997**	0.997
E	0.865	0.590	0.914	0.922	**0.934**

Table 11.1 Performance of each class of policy on each problem, relative to the optimal posterior solution (from Powell & Meisel (2016)). Bold indicates the best performer.

The table shows that *each* of the five policies works best on one of the five problems. Of course, the problems were designed so that this was the case, but this illustrates that any of the policies can be best, even on a single problem class, just by modifying the data. For example, a deterministic lookahead works best when the forecast is quite good. A VFA-based strategy works best on problems that are very time-dependent, with a high degree of uncertainty (that is, the forecasts are poor). The hybrid CFA-based policy works best when the forecast is uncertain, but adds value.

11.9 DESIGNING A POLICY

Given the choice of policies, the question naturally arises, how do we design a policy that is best for a particular problem? Not surprisingly, it depends on the characteristics of the problem, constraints on computation time, and the complexity of the algorithm. Below we summarize different types of problems, and provide a sample of a policy that appears to be well suited to the application, largely based on our own experiences with real problems.

Policy function approximations

A utility would like to know the value of a battery that can store electricity when prices are low and release them when prices are high. The price process is highly volatile, with a modest daily cycle. The utility needs a simple policy that is easy to implement in software. The utility chose a policy where we fix two prices, and store when prices are below the lower level and release when prices are above the higher level. This requires optimizing these two price points. A different policy might involve storing at a certain time of day, and releasing at another time of day, to capture the daily cycle.

The PFA is a natural choice because we understand the structure of the policy. It seems clear (and supporting research proves that this is the case) that a "buy low, sell high" policy is optimal. In many cases, the structure of a PFA seems apparent, but lacks any proof of optimality, and may not be optimal, but likely works quite well.

Even when a PFA seems apparent, two problem characteristics may limits its accuracy:

- Time dependency - It may easily be the case that the parameters of our PFA (e.g. the points at which we buy and sell electricity) are time dependent. It is relatively easy to optimize over two parameters. If there are 100 time periods, it is an entirely different matter to optimize over 200 parameters.

- State dependency - Our policy may depend on other state variables such as weather (in our energy storage attribute). In a health application, we may be able to design a PFA to determine the dosage of a medication to lower blood sugar. For example, we may be able to design a simple linear (or piecewise linear) function relating the dosage to the level of blood sugar. But the choice of drug (there are dozens) may depend on patient attributes, and we may need a different PFA for each drug (and perhaps even reflecting different patient attributes). Now our PFA is much more complex.

Cost function approximation

Cost function approximations may easily be the most widely used class of policy in real applications, although as a class they have been largely ignored by the research literature. Pure CFAs arise whenever we have a policy that involves finding the best within a set (or a feasible region), where we are maximizing (or minimizing) some function that does not explicitly capture the impact of decisions now on the future. Since decisions (almost always) do have an impact on the future, CFAs have to be tuned to work well over time.

We first saw CFAs used very effectively in pure learning problems in chapter 7. For example, the interval estimation policy

$$X^{IE}(S^n|\theta^{IE}) = \arg\max_x \left(\bar{\mu}^n_x + \theta^{IE}\bar{\sigma}^n_x\right),$$

which trades off exploitation (by maximizing over $\bar{\mu}_x^n$ which is our estimate of how well choice x might work) and exploration (by maximizing over $\bar{\sigma}_x^n$ which is the standard deviation of our estimate $\bar{\mu}_x^n$). The weight that we put on $\bar{\sigma}_x^n$ relative to $\bar{\mu}_x^n$, given by θ^{IE}, has to be tuned.

CFAs are useful when we have an intuitive idea of how to handle uncertainty. Consider the problem of deciding on a time to leave for work for your job in a dense city. Your navigation system tells you that the trip will take 37 minutes, so you add 10 minutes to be safe. After following this strategy for a week, you arrive late one day because of an unexpected delay, so you increase your buffer to 15 minutes. This is a form of CFA which is searching for the best path, and then adding a tunable buffer to account for uncertainty.

CFAs are also well-suited to complex, high dimensional problems such as scheduling an airline. In this setting, we would solve a large, deterministic integer program to schedule planes and crews, but we have to deal with the uncertainty of flight times due to congestion and weather delays. The airline adds a buffer which may depend on both the origin and destination, but also the time of day. This buffer might be based on a dataset where the airline chooses a buffer so that the flight should be on-time θ percent of the time. The airline will then monitor network-wide on-time performance and feedback from customers to help it tune θ.

Value function approximations

As with PFAs and CFAs, value function approximations are best suited when approximating the value of being in a state seems straightforward. As always, this is problem dependent. Some examples where this seems relatively easy to do include:

A trucking problem A truckload carrier needs to figure out which driver to assign to a load. Driver are characterized by their home, their current location, their equipment type, and how long they have been away from home. Let R_{ta} be the number of drivers with attribute vector a capturing these attributes. This means we need to estimate $\bar{v}_{ta}$ which is the approximate value of a driver with attribute a. The value of the resource vector $R_t = (R_{ta})_{a\in\mathcal{A}}$ might be reasonable approximated by

$$V_t(R_t) \approx \overline{V}_t(R_t) = \sum_{a\in\mathcal{A}} \bar{v}_{ta} R_{ta}.$$

Inventory problems There are many problems where R_t is a scalar describing the inventory of product for sale, blood supplies, energy in a battery, or cash in a mutual fund.

Routing on a graph We are at a node i and need to determine which link (i, j) to go to, where traversing a link incurs a random cost $\hat{c}_{ij}$ which is revealed after we move from i to j. We need to learn the value $\bar{v}_i$ of being at each node to make the best decision.

These are all examples of problems where value function approximations tend to work well. We now list some problems where they do not work well:

A trucking problem Consider the trucking problem above, but now we have to keep track of both the number of drivers R_{ta} of each type, but also loads, where L_{tb} is the number of loads with attribute vector b (origin, destination, equipment requirements). Loads that are not served at time t are held until time $t + 1$. Now we need to approximate a value function $V_t(R_t, D_t)$. The value of a driver in a region depends on the number and types of loads in that region, so these are tightly coupled.

Inventory problems Imagine that while managing our inventory R_t we have to consider other dynamic data. For example, if R_t is how much energy is in the battery, we might also have to keep track of the current and previous prices of energy, the temperature, and the demand for energy. Another complicate might be the availability of forecasts (for demands or any of the exogenous data), where the forecast may be of high or low quality.

Routing on a graph Assume that when we arrive at node i that we get to see the actual cost $\hat{c}_{ij}$ before we traverse a link, so we can use this information when traversing the link.

Direct lookahead policies

Direct lookahead policies are what we turn to when we are unable to identify suitable approximations that are required when designing PFAs, CFAs and VFAs. This is a reason why this strategy is popular in video games, chess and Go. These are problems with relatively small action spaces, but very complex state spaces. For example, putting a bishop at a particular place on a chess board depends on all the remaining pieces on the board.

There are two important strate

Deterministic lookahead Sometimes known as model predictive control or a rolling/receding horizon procedure, a deterministic lookahead is often the first policy that many will try. A deterministic lookahead is typically used when the environment is nonstationary (e.g. time of day or seasonal variations), or nonuniform. For example, when you use a navigation system to find the shortest path to a destination, while using point estimates for the time over each leg of the network, this is a deterministic lookahead (a transportation network is highly nonuniform). This is the only option in this setting because there is no natural PFA, CFA or VFA (other than our earlier example of solving a shortest path problem and adding a buffer, which is a form of lookahead-CFA hybrid).

Deterministic lookaheads can often be good approximations even in the presence of uncertainty. For example, it works quite well in planning paths to a destination even though travel times over each leg of the network are random. One major class where a deterministic lookahead struggles is when buying and selling assets in the presence of highly variable prices.

Sampled lookaheads If a deterministic lookahead (including a modified deterministic lookahead) does not seem as if it will work, then a popular strategy is to represent the future using a finite (and not very large) number of "scenarios" (as they are typically referred to).

A popular application for sampled lookahead models is in regions where a significant amount of electricity generation is coming from hydroelectric power, requiring a utility to manage the flows of water into and out of the reservoirs. There can be tremendous uncertainty about the rainfall over the course of a season, and as a result the utility would like to make decisions now while accounting for the different possible outcomes. Stochastic programming enumerates all the decisions over the entire year, for each possible scenario (while ensuring that only one decision is made now). The stochastic program ensures that all constraints will be satisfied for each scenario.

We return to this approach in greater detail in chapter 20.

Discussion

These examples raise a series of questions that should be asked when choosing the structure of a policy:

- Will a myopic policy solve your problem? If not, is it at least a good starting point?
- Does the problem have structure that suggests a simple and natural decision rule? If there is an "obvious" policy (e.g. replenish inventory when it gets too low), then more sophisticated algorithms based on value function approximations are likely to struggle. Exploiting structure always helps.
- Is the problem fairly stationary, or highly nonstationary? Nonstationary problems (e.g. responding to hourly demand or daily water levels) mean that you need a policy that depends on time. Rolling horizon problems can work well if the level of uncertainty is low relative to the predictable variability. It is hard to produce policy function approximations where the parameters vary by time period.
- If you think about approximating the value of being in a state, does this appear to be a relatively simple problem? If the value function is going to be very complex, it will be hard to approximate, making value function approximations hard to use. But if it is not too complex, value function approximations may be a very effective strategy.

Unless you are pursuing an algorithm as an intellectual exercise, it is best to focus on your problem and choose the method that is best suited to the application. For more complex problems, be prepared to use a hybrid strategy. For example, rolling horizon procedures may be combined with adjustments that depend on tunable parameters (a form of policy function approximation). You might use a lookahead policy using a decision tree combined with a simple value function approximation to help reduce the size of the tree.

11.10 POLICY EVALUATION, TUNING AND SEARCH

We are now going to walk through the process of how we would compute the value of these policies which helps to make the expectations more transparent. It helps to have a motivating application to illustrate the ideas. Imagine that we want to find a policy to control the buying and selling of energy to/from the grid. Let $x = (x^{buy}, x^{sell})$ be a choice of when to buy and sell electricity, where $\mathcal{X} = \{x^1, \ldots, x^K\}$ is a set of possible buy-sell limits. With a little creativity we can make our problem state-independent (e.g. we do not know the price when we make a buy-sell decision) or state-dependent (we do know the price or we have to manage the charge level of the battery).

We need a policy for deciding which x_t to test next (at time t), given our knowledge about $\mathbb{E}\overline{F}^{\pi}(\theta)$. Our "policy" can be any adaptive learning procedure, which might be derivative-based (stochastic gradient algorithm, SDDP) or derivative-free (upper confidence bounding, Q-learning, approximate dynamic programming). To keep things simple, we are going to illustrate the concepts using an interval estimation policy given by

$$X^{IE}(S_t|\theta^{IE}) = \arg\max_{x \in \mathcal{X}} \left(\overline{F}_t(x) + \theta^{IE}\bar{\sigma}_t(x)\right), \tag{11.36}$$

where $S_t = (\overline{F}_t(x), \bar{\sigma}_t(x))_{x \in \mathcal{X}}$. We can evaluate our policy using either the cumulative reward (equations (9.40) and (9.42)), or final reward (equations (9.38) or (9.44)).

We begin by reviewing how to approximate a policy using simulation. We then discuss the challenge of tuning a policy within a particular class. We end by addressing the problem of searching across classes.

11.10.1 Policy evaluation

In chapter 9, section 9.10, we described four problem classes based on whether we were maximizing final reward or cumulative reward, and whether we were solving state-independent functions (learning problems), or state-dependent problems. These problems were given in section 9.10 with a summary of how to compute them in section 9.10. We refer the reader back to these sections for more complete discussions but review the objectives and how to simulate them below.

Throughout our presentation, we let ω represent a sample of $W^1, \ldots, W^N$ which we sample while we are learning a decision $x^{\pi,N}$ (or policy $X^{imp}(S_t)$) in an offline, final-reward problem, or for the complete learning and testing in an online, cumulative-reward problem. While following sample path ω, our states evolve according to

$$S^{n+1}(\omega) = S^M(S^n(\omega), X^\pi(S^n(\omega)), W^{n+1}(\omega))$$

if we are indexing by iterations, or

$$S_{t+1}(\omega) = S^M(S_t(\omega), X^\pi(S_t(\omega)), W_{t+1}(\omega))$$

if we are indexing by time.

When we are in an offline setting, after we have learned our solution or policy, we then have to test it, which we do by sampling our testing random variable $\widehat{W}$. We let ψ represent a sample realization of $\widehat{W}$.

Class 1) State-independent, final reward:

$$\begin{aligned} \max_\pi F^\pi &= \mathbb{E}\{F(x^{\pi,N}, \widehat{W})|S^0\} \\ &= \mathbb{E}_{S^0}\mathbb{E}_{W^1,\ldots,W^N|S^0}\mathbb{E}_{\widehat{W}|S^0}F(x^{\pi,N}, \widehat{W}). \end{aligned}$$

We then simulate F^π using

$$F^\pi(\theta|\omega, \psi) = F(x^{\pi,N}(\theta|\omega), \widehat{W}(\psi)).$$

Finally we approximate the expectation by averaging using

$$\overline{F}^\pi(\theta) = \frac{1}{K}\frac{1}{L}\sum_{k=1}^{K}\sum_{\ell=1}^{L} F^\pi(\theta|\omega^k, \psi^\ell).$$

Class 2) State-independent, cumulative reward reward:

$$\begin{aligned}\max_{\pi} F^{\pi} &= \mathbb{E}\left\{\sum_{n=0}^{N-1} F(X^{\pi}(S^n), W^{n+1})|S^0\right\} \\ &= \mathbb{E}_{S^0}\mathbb{E}_{W^1,\ldots,W^N|S^0} \sum_{n=0}^{N-1} F(X^{\pi}(S^n), W^{n+1}).\end{aligned}$$

We then simulate F^{π} using

$$F^{\pi}(\theta|\omega) = \sum_{n=0}^{N-1} F(X^{\pi}(S^n(\omega)|\theta), W^{n+1}(\omega)).$$

Finally we approximate the expectation by averaging using

$$\overline{F}^{\pi}(\theta) = \frac{1}{K}\sum_{k=1}^{K} F^{\pi}(\theta|\omega^k).$$

Class 3) State-dependent, cumulative reward:

$$\begin{aligned}\max_{\pi} F^{\pi} &= \mathbb{E}\left\{\sum_{t=0}^{T} C(S_t, X^{\pi}(S_t), W_{t+1})|S_0\right\} \\ &= \mathbb{E}_{S_0}\mathbb{E}_{W_1,\ldots,W_T|S_0} \left\{\sum_{t=0}^{T} C(S_t, X^{\pi}(S_t), W_{t+1})|S_0\right\}.\end{aligned}$$

We then simulate F^{π} using

$$F^{\pi}(\theta|\omega) = \sum_{t=0}^{T-1} C(S_t(\omega), X^{\pi}(S_t(\omega)|\theta), W_{t+1}(\omega)).$$

We then average over sample paths to obtain

$$\overline{F}^{\pi}(\theta) = \frac{1}{K}\sum_{k=1}^{K} F^{\pi}(\theta|\omega^k).$$

Class 4) State-dependent, final reward:

$$\begin{aligned}\max_{\pi^{lrn}} F^{\pi^{lrn}} &= \mathbb{E}\{C(S, X^{\pi^{impl}}(S|\theta^{imp}), \widehat{W})|S^0\} \\ &= \mathbb{E}_{S^0}\mathbb{E}^{\pi^{imp}}_{\left((W_t^n)_{t=0}^T\right)_{n=0}^N|S^0} \left(\mathbb{E}^{\pi^{imp}}_{(\widehat{W}_t)_{t=0}^T|S^0} \frac{1}{T}\sum_{t=0}^{T-1} C(S_t, X^{\pi^{impl}}(S_t|\theta^{imp}), \widehat{W}_{t+1})\right).\end{aligned}$$

We then simulate F^{π} using

$$F^{\pi}(\theta^{lrn}|\omega, \psi) = \sum_{t=0}^{T} C(S_t(\omega), X^{\pi^{imp}}(S_t(\omega)|\theta^{imp}), \widehat{W}_{t+1}(\psi)).$$

We then average over sample paths to obtain

$$\overline{F}^{\pi}(\theta^{lrn}) = \frac{1}{K}\frac{1}{L}\sum_{k=1}^{K}\sum_{\ell=1}^{L} F^{\pi}(\theta^{lrn}|\omega^k, \psi^{\ell}).$$

For each of these classes, we assume that π determines the structure of the policy (or the learning policy π^{lrn}), and that θ (or θ^{lrn}) represents any tunable parameters in a set of parameters determined by π. We next address the problem of tuning θ, which is typically one or more continuous parameters (these may be discrete but numerical values).

11.10.2 Parameter tuning

Parameter tuning in policy search is its own stochastic optimization problem to find a policy (or algorithm) to solve a stochastic optimization problem which we can write as

$$\max_{\theta} F^{\pi}(\theta). \tag{11.37}$$

Since $F^{\pi}(\theta)$ involves an expectation we cannot compute we typically are solving

$$\max_{\theta} \overline{F}^{\pi}(\theta). \tag{11.38}$$

Independent of which class of problem produces our function $F^{\pi}(\theta)$ (or $\overline{F}^{\pi}(\theta)$), we need to find a (possibly vector-valued) parameter θ that controls our implementation policy (or how we find our implementation policy).

If we can take derivatives of $F^{\pi}(\theta)$ (or our sampled approximation $\overline{F}^{\pi}(\theta)$), we might find θ using a stochastic gradient algorithm

$$\theta^{n+1} = \theta^n + \alpha_n \nabla_{\theta}\overline{F}^{\pi,n+1}(\theta^n),$$

where $\overline{F}^{\pi,n+1}(\theta^n)$ is the $n+1st$ sampled estimate of $\overline{F}^{\pi}(\theta^n)$. Assume that $A^{\pi^{step}}(S^n|\theta^{\alpha})$ is our stepsize policy parameterized by θ^{α}. If we apply this stepsize policy, assume that this then produces an estimate of the tunable parameter in $X^{\pi}(S|\theta)$ after N iterations of $\theta = \theta^{\pi^{step},N}$.

If we do not have access to derivatives, we might run a search among the set $\Theta = \{\theta_1, \ldots, \theta_K\}$. For example, we could use an interval estimation policy where we compute

$$\Theta^{\pi^{IE}}(S^n|\theta^{IE}) = \arg\max_{\theta\in\Theta}\left(\overline{F}^{\pi,n}(\theta) + \theta^{IE}\bar{\sigma}^n_{\theta}\right),$$

where $\bar{\sigma}^n_{\theta}$ is the standard deviation of our estimate $\overline{F}^{\pi,n}(\theta)$, where we draw on our statistical modeling presented in chapter 3.

Now we have to choose the best search policy π^{search} (parameterized by θ^{search}), whether it is the stepsize policy (which means we have to choose the type of stepsize rule and any tunable parameters) or a type of derivative-free learning rule (such as interval estimation) along with its tunable parameters (such as θ^{IE}). We need search policies/algorithms to learn implementation decisions $x^{\pi,N}$, implementation policies $X^{\pi^{imp}}(S)$, or online learning policies $X^{\pi}(S)$ (for classes (2) and (3)).

In practice, finding and tuning search algorithms such as the stochastic gradient algorithm or the interval estimation policy tends to be fairly ad hoc. Formal analysis of search algorithms tends to fall in one of three categories:

Asymptotic convergence Probably the most standard result for an algorithm is a proof that the solution will asymptotically approach the optimal solution (that is, as the number of iterations $N \to \infty$). The criticism of asymptotic convergence is that it says nothing about rate of convergence, which means it is not telling us anything about the quality of the solution after N iterations.

Finite-time bounds These are results that suggest that the quality of the solution after N iterations is within some limit. These bounds tend to be quite weak, and often feature unknown coefficients.

Asymptotic rate of convergence It is often possible to provide high quality estimates of the rate of convergence, but only when the solution is in the vicinity of the optimal.

The holy grail of theoretical analysis of algorithms is tight bounds for the performance after n iterations. These are rare, and are limited to very simple problems. For this reason, empirical analysis of algorithms remains an important part of the design and analysis of search algorithms. Frustratingly, the performance of a search algorithm on one dataset may not guarantee good performance on a different dataset, even for the same problem class.

In practice, "optimizing" the search policy/algorithm is often limited to a small number of experiments, possibly (but not always) testing different classes of search policies (such as different classes of stepsize rules), and a small number of values for the search parameters θ^{step} or θ^{IE}. As of this writing, we are not aware of any work that formally addresses the problem of finding optimal search procedures.

This raises the question: if we are looking for the best implementation decision/policy, how good does the search policy/algorithm need to be to find an implementation decision/policy?

Just as a weak algorithm for a deterministic optimization problem can produce a poor solution, a weak search policy/algorithm can produce a poor implementation decision (or policy). In fact, the results can be quite poor. For example, in chapter 14 we are going to describe a class of algorithms for discrete state, discrete action sequential decision problems where we can find provably optimal policies (we cannot have too many states or actions). We then describe adaptive learning algorithms for solving larger versions of these same problems. If we apply these adaptive learning algorithms to problems that we can solve optimally, we get a sense of how well our approximate policies perform against optimal policies. It is not hard to find applications where the approximate policies are quite poor.

11.10.3 Searching across policy classes

The previous section focused on tuning the parameters of a particular policy class. What about searching across policy classes? If the number of classes being tested is small, a reasonable strategy is to analyze each of the policy classes and choose the best one. Of course, we can do better, since this is basically a search over discrete choices.

Rather than evaluate each policy class in depth, we can do a partial evaluation, just as we would examine an unknown function. This introduces the issue of having to optimize over a set of parameters in order to evaluate a particular search policy/algorithm. If this is easy, then finding the best search policy/algorithm may not be as critical. However, imagine finding the best search policy for a problem where derivatives are not available, and function evaluations take several hours (or a day).

11.11 BIBLIOGRAPHIC NOTES

The goal of this chapter is to organize the diverse policies that have been suggested in the ADP and RL communities into a more compact framework. In the process, we are challenging commonly held assumptions, for example, that "approximate dynamic programming" always means that we are approximating value functions, even if this is one of the most popular strategies. Lookahead policies, and policy function approximations are effective strategies for certain problem classes.

Section 11.5 - Lookahead policies have been widely used in engineering practice in operations research under the name of rolling horizon procedure, and in computer science as a receding horizon procedure, and in engineering under the name model predictive control (Camacho & Bordons (2003)). Decision-trees are similarly a widely used strategy for which there are many references. Roll-out heuristics were introduced by **?** and Bertsekas & Castanon (1999). Stochastic programming, which combines uncertainty with multi-dimensional decision vectors, is reviewed in Birge & Louveaux (1997) and Shapiro (2003), among others. Secomandi (2008) studies the effect of reoptimization on rolling horizon procedures as they adapt to new information.

Section 11.2 - While there are over 1,000 papers which refer to "value function approximation" in the literature (as of this writing), there were only a few dozen papers using "policy function approximation." However, this is a term that we feel deserves more widespread use as it highlights the symmetry between the two strategies.

Section 11.4 - Making decisions which depend on an approximation of the value of being in a state has defined approximation dynamic programming since Bellman & Kalaba (1959*a*).

PROBLEMS

11.1 What is the difference between a stationary policy, a deterministic nonstationary policy, and an adaptive policy?

11.2 Following is a list of how decisions are made in specific situations. For each, classify the decision function in terms of which of the four fundamental classes of policies are being used. If a policy function approximation or value function approximation is used, identify which functional class is being used:

- If the temperature is below 40 degrees F when I wake up, I put on a winter coat. If it is above 40 but less than 55, I will wear a light jacket. Above 55, I do not wear any jacket.

- When I get in my car, I use the navigation system to compute the path I should use to get to my destination.

- To determine which coal plants, natural gas plants and nuclear power plants to use tomorrow, a grid operator solves an integer program that plans over the next 24 hours which generators should be turned on or off, and when. This plan is then used to notify the plants who will be in operation tomorrow.

- A chess player makes a move based on her prior experience of the probability of winning from a particular board position.
- A stock broker is watching a stock rise from \$22 per share up to \$36 per share. After hitting \$36, the broker decides to hold on to the stock for a few more days because of the feeling that the stock might still go up.
- A utility has to plan water flows from one reservoir to the next, while ensuring that a host of legal restrictions will be satisfied. The problem can be formulated as a linear program which enforces these constraints. The utility uses a forecast of rainfalls over the next 12 months to determine what it should do right now.
- The utility now decides to capture uncertainties in the rainfall by modeling 20 different scenarios of what the rainfall might be on a month-by-month basis over the next year.
- A mutual fund has to decide how much cash to keep on hand. The mutual fund uses the rule of keeping enough cash to cover total redemptions over the last 5 days.
- A company is planning sales of TVs over the Christmas season. It produces a projection of the demand on a week by week basis, but does not want to end the season with zero inventories, so the company adds a function that provides positive value for up to 20 TVs.
- A wind farm has to make commitments of how much energy it can provide tomorrow. The wind farm creates a forecast, including an estimate of the expected amount of wind and the standard deviation of the error. The operator then makes an energy commitment so that there is an 80 percent probability that he will be able to make the commitment.

11.3 Consider two policies:

$$X^{\pi^A}(S_t|\theta) = \arg\max_{x_t}\left(C(S_t,x_t) + \sum_{f\in\mathcal{F}}\theta_f\phi_f(S_t)\right), \tag{11.39}$$

and

$$X^{\pi^B}(S_t|\theta) = \arg\max_{x_t}\left(C(S_t,x_t) + \sum_{f\in\mathcal{F}}\theta_f\phi_f(S_t)\right). \tag{11.40}$$

In the case of the policy π^A in equation (11.39), we search for the parameter vector θ by solving

$$\max_\theta \mathbb{E}\sum_{t=0}^{T} C(S_t, X^{\pi^A}(S_t|\theta)). \tag{11.41}$$

In the case of policy π^B, we wish to find θ so that

$$\sum_{f\in\mathcal{F}}\theta_f\phi_f(S_t) \approx \mathbb{E}\sum_{t'=t}^{T} C(S_t, X^{\pi^B}(S_t|\theta)). \tag{11.42}$$

a) (5 points) Classify policies π^A and π^B among the four classes of policies.

b) (5 points) Can we expect that the value θ^A that optimizes (11.41) would be approximately equal to the value θ^B that solves equation (11.42)?

11.4 Below is a list of problems with a proposed method for making decisions. Classify each method based on the four classes of policies (you may decide that a method is a hybrid of more than one class).

a) (3 points) You use Google maps to find the best path to your destination.

b) (3 points) You are managing a shuttle service between the mainland and a small resort island. You decide to dispatch the shuttle as soon as you reach a minimum number of people, or when the wait time of the first person to board exceeds a particular amount.

c) (3 points) An airline optimizes its schedule over a month using schedule slack to protect against potential delays.

d) (3 points) Upper confidence bounding policies for performing sequential learning (these were introduced in chapter 7).

e) (3 points) A computer program for playing chess using a point system to evaluate the value of each piece that has not yet been captured. Assume it chooses the move that leaves it with the highest number of points after one move.

f) (3 points) Imagine an improved computer program that enumerates all possible chess moves after three moves, and then applies its point system.

g) (3 points) Thompson sampling for sequential learning (also introduced in chapter 7).

11.5 You are the owner of a racing team, and you have to decide whether to keep going with your current driver or to stop and consider a new driver. The decision after each race is to stay with your driver or stop (and switch). The only outcome you care about is whether your driver won or not.

a) Formulate the problem as a decision tree over three races (we index these races as 0, 1 and 2).

b) In equation (11.17), we write our optimal policy as

$$X_t^*(S_t) = \arg\max_{x_t} \left(C(S_t, x_t) + \mathbb{E}\left\{ \max_{\pi} \mathbb{E}\left\{ \sum_{t'=t+1}^{T} C(S_{t'}, X_{t'}^{\pi}(S_{t'})) \middle| S_{t+1} \right\} |S_t, x_t \right\} \right). \quad (11.43)$$

Letting $t = 0$ where we face one of two actions (stay with current driver or replace), fully enumerate all the policies we may consider for $t = 1, 2$.

c) The outer expectation $\mathbb{E}$ in (11.43) is over which random variable(s)?

d) The inner expectation $\mathbb{E}$ in (11.43) is over which random variable(s)?

11.6 Earlier we considered the problem of assigning a resource i to a task j. If the task is not covered at time t, we hold it in the hopes that we can complete it in the future. We would like to give tasks that have been delayed more higher priority, so instead of just just maximizing the contribution c_{ij}, we add in a bonus that increases with how long the task has been delayed, giving us the modified contribution

$$c^{\pi}_{tij}(\theta) = c_{ij} + \theta_0 e^{-\theta_1(\tau_j - t)}.$$

Now imagine using this contribution function, but optimizing over a time horizon T using forecasts of tasks that might arrive in the future. Would solving this problem, using $c^{\pi}_{tij}(\theta)$ as the contribution for covering task j using resource i at time t, give you the behavior that you want?

11.7 We wish to use Q-learning to solve the problem of deciding whether to continue playing a game where you win \$1 if you flip a coin and see heads, and lose \$1 if you see tails. Using a stepsize $\alpha = \frac{\theta}{\theta+n}$, implement the Q-learning algorithm in equations (11.12) and (11.13). Initialize your estimates $\bar{Q}(s,a) = 0$, and run 1000 of the algorithm using $\theta = 1$, 10, 100 and 1000. Plot Q^n for each of the three values of θ, and discuss the choice you would make if your budget was $N = 50$, 100 or 1000.

PART IV - POLICY SEARCH

Policy search is a strategy where we define a class of functions that determine a decision, and then search for the best function within that class. Policy search arises in two settings:

- Optimizing an analytical function - We refer to these as *policy function approximations*, where we search over a set of parameters to find the function that produces the best results. PFAs are typically limited to scalar actions or low-dimensional controls.

- Parametric cost function approximations - This strategy is used for high-dimensional constrained problems, such as might arise when planning energy generation over a grid, scheduling an airline or managing a supply chain. This strategy typically requires a parametric modification of the cost function or, more frequently, a modified set of constraints.

Policy search applied to finding analytical policy function approximations has been widely studied in the academic literature. There are close parallels between policy search and classical machine learning; the difference is primarily in the metric being minimized. Finding the best parametric cost function approximation has been ignored by the academic research community, but is widely used in industry, where it is typically characterized as a "deterministic model."

CHAPTER 12

POLICY FUNCTION APPROXIMATIONS AND POLICY SEARCH

A policy function approximation is any analytical function mapping a state to an action. We use the term "policy function approximation" to distinguish this class from a generic "policy" which covers any mapping from state to action. These may come in any one of three forms: lookup table (where we map a discrete state to a discrete action), parametric functions, and nonparametric (or more likely, locally parametric) functions. What we exclude are the other three classes of functions which all require solving an optimization problem to determine an action.

Most of our attention will be devoted to parametric functions that are characterized by a small set of parameters. Some examples are listed below.

■ **EXAMPLE 12.1**

A basic inventory policy is to order product when the inventory goes below some value θ^L where we order up to some upper value θ^U. If S_t is the inventory level, this policy might be written

$$X^\pi(S_t|\theta) = \begin{cases} \theta^U - S_t & \text{If } S_t < \theta^L, \\ 0 & \text{Otherwise.} \end{cases}$$

■ **EXAMPLE 12.2**

Stochastic Optimization and Learning. By Warren B. Powell

If S_t is a scalar variable giving, for example, the rainfall over the last week, we might set a policy for releasing water from a reservoir using

$$X^\pi(S_t|\theta) = \theta_0 + \theta_1 S_t + \theta_2 S_t^2.$$

■ EXAMPLE 12.3

The outflow u_t of a water reservoir is given by a piecewise linear function of the reservoir level R_t according to:

$$U^\pi(S_t|\theta) = \begin{cases} 0 & R_t < R^{min}, \\ \theta_1 & 0 \times (R^{max} - R^{min}) \leq R_t - R^{min} \leq .2(R^{max} - R^{min}), \\ \theta_2 & .2 \times (R^{max} - R^{min}) \leq R_t - R^{min} \leq .4(R^{max} - R^{min}), \\ \theta_3 & .4 \times (R^{max} - R^{min}) \leq R_t - R^{min} \leq .6(R^{max} - R^{min}), \\ \theta_4 & .6 \times (R^{max} - R^{min}) \leq R_t - R^{min} \leq .8(R^{max} - R^{min}), \\ \theta_5 & .8 \times (R^{max} - R^{min}) \leq R_t - R^{min} \leq 1.0(R^{max} - R^{min}), \\ \theta^{max} & R_t > R^{max}. \end{cases}$$

where we would expect $\theta_{i+1} \geq \theta_i$.

■ EXAMPLE 12.4

A popular strategy in the engineering community is to train a policy $U^\pi(S_t|\theta)$ using a neural network which is characterized by a set of layers and a set of weights that are captured by θ (we provided a brief description of neural networks in section 3.9.3) which takes as input a state variable S_t and outputs a control u_t.

Each of these examples involves a policy parameterized by a parameter vector θ. In principle, we can represent a lookup table using this notation where there is a parameter θ_s for each discrete state s. However, most problems exhibit a large (potentially infinite) number of states, which translates to an equally large (and potentially infinite) number of parameters. As of this writing, we do not have practical algorithms for searching over parameter spaces that are this large.

We begin by describing different classes of policies where we focus on policies that have attracted some attention in the literature. Afterward, we turn our attention to the much harder of optimizing these parameters. This is accomplished using the basic formulation

$$\max_{\theta \in \Theta^\pi} \mathbb{E} \sum_{t=0}^{T} C(S_t, X^\pi(S_t|\theta)), \tag{12.1}$$

where $S_{t+1} = S^M(S_t, X^\pi(S_t|\theta), W_{t+1})$, and where the expectation is over the different possible sequences $W_1, \ldots, W_T$. The search is over some space Θ^π that corresponds to the class of policy we have chosen. As we show, this disarmingly simple formulation can be quite hard to solve.

12.1 CLASSES OF POLICY FUNCTION APPROXIMATIONS

Policy function approximations are almost exclusively some form of parametric or locally parametric function. We are going to begin by illustrating a number of these. One attraction of parametric policies is that it is possible to take derivatives of them with respect to their parameters. For this reason, we then transition to finding the gradient of our objective function which is computed by simulating the policy.

12.1.1 Lookup table policies

A lookup table policy is a function where for a particular discrete state s we return a discrete action $x = X^\pi(s)$. This means we have one parameter (an action) for each state. We exclude from this class any policies that can be parameterized by a smaller number of parameters. Lookup tables are relatively common in practice, since they are easy to understand. For example, the Transportation Safety Administration (TSA) has specific rules that determine when and how a passenger should be searched. Call-in centers use specific rules to govern how a call should be routed. Expert chess players are able to look at a board (in the initial stages of a game) and know exactly what move to make. A doctor will often take a set of symptoms and patient characteristics to determine the right treatment.

Lookup tables are easy to understand, and easy to enforce. But in practice, they can be very hard to optimize since there is a value (the action) for each state. So, if we have $|\mathcal{S}| = 1000$ states, searching directly for the best policy would mean searching over a 1000-dimensional parameter space (the action to be taken in each state).

One attraction of lookup table policies is that they are very easy to compute in production (imagine a real-time setting where decisions have to be made with exceptional speed). In business, lookup table policies are widely used where they are known as business rules (although these rules may often be parameterized). In practice these rules are never optimized; the rules are specified by human judgment and then left alone, but that does not mean that we could not consider optimizing them.

12.1.2 Boltzmann policies for discrete actions

A Boltzmann policy chooses a discrete action $x \in \mathcal{X}_s$ according to the probability distribution

$$f(x|s,\theta) = \frac{e^{\theta \bar{C}(s,x)}}{\sum_{x' \in \mathcal{X}} e^{\theta \bar{C}(s,x)}}.$$

where $\bar{C}(s,x)$ is some sort of contribution to be maximized. This could be our estimate of a function $\mathbb{E}F(x,W)$ as we did in chapter 7, or an estimate of the one-step contribution plus a downstream value, as in

$$\bar{C}(S^n,x) = C(S^n,x) + \mathbb{E}\{\overline{V}^n(S^{n+1})|S^n,x\},$$

where $\overline{V}^n(S)$ is our current estimate of the value of being in state S.

Let $F(a|S^n,\theta)$ be the cumulative distribution of our probabilities

$$F(x|s,\theta) = \sum_{x' \leq x} f(x'|s,\theta).$$

Let $U \in [0,1]$ be a uniformly distributed random number. Our policy $X^\pi(s|\theta)$ could be written

$$X^\pi(s|\theta) = \arg\max_x \{F(x|s,\theta)|F(x|s,\theta) \leq U\}.$$

This is an example of a so-called "stochastic policy," but we handle it just as we would any other policy.

12.1.3 Affine policies

An "affine policy" is any policy that is linear in the unknown parameters. Thus, an affine policy might be of the form

$$U^{\pi}(S_t|\theta) = \theta_0 + \theta_1\phi_1(S_t) + \theta_2\phi_2(S_t).$$

We first saw affine policies in chapter 4 when we presented the linear quadratic control problem which, in our notation, is given by

$$\min_{\pi} \mathbb{E} \sum_{t=0}^{T} \big((S_t)^T Q_t S_t + (x_t)^T R_t x_t\big). \tag{12.2}$$

After considerable algebra, it is possible to show that the optimal policy $X_t^*(S_t)$ is given by

$$X_t^*(S_t) = K_t S_t,$$

where K_t is a suitably dimensioned matrix that is a function of the matrices Q_t and R_t. Of course, we assume that S_t and x_t are continuous vectors. Thus, $X^*(S_t)$ is a linear function of S_t with coefficients determined by the matrix K_t.

This result requires that the objective function be quadratic (or a mixture of quadratic and linear) functions of the state S_t and control x_t. It also requires that the problem be unconstrained, which can be a reasonable starting point for many problems in robotic controls where forces x_t can be positive or negative, and where some constraints (such as the maximum force) would simply not be binding.

We can often handle violations of the key assumptions by using the optimal policy for a simplified or relaxed version of the problem as a starting point.

12.1.4 Locally linear policies

A surprisingly powerful strategy for many problems with continuous states and actions is to assume locally linear responses. For example, S_t may capture the level of a reservoir, or the current speed and altitude of a helicopter. The control x_t could be the rate at which water is released from the reservoir, or the forces applied to the helicopter. Assume that we use our understanding of the problem to create a family of regions $\mathcal{S}_1, \ldots, \mathcal{S}_I$, which are most likely going to be a set of rectangular regions (or intervals if there is only one dimension). We might then create a family of linear (affine) policies of the form

$$X_i^{\pi}(S_t|\theta) = \theta_{i0} + \theta_{i1}\phi_1(S_t) + \theta_{i2}\phi_2(S_t),$$

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

This approach has been found to be very effective in some classes of control problems. In practice, the regions $\mathcal{S}_i$ are designed by someone with an understanding of the physics of the problem. Further, instead of tuning one vector θ, we have to tune $\theta_1, \ldots, \theta_I$. While this can represent a laboratory challenge, the approach can work quite well, and offers the important feature that they can be computed extremely quickly.

12.1.5 Monotone policies

There are a number of problems where the decision increases, or decreases, with the state variable. If the state variable is multidimensional, then the decision (which we assume is scalar) increases, or decreases, with *each* dimension of the state variable. Policies with this structure are known as *monotone policies*. Some examples include:

- There are a number of problems with binary actions that can be modeled as $x \in \{0,1\}$. For example
  - We may hold a stock ($x_t = 0$) or sell ($x_t = 1$) if the price p_t falls below a smoothed estimate $\bar{p}_t$ which we compute using

    $$\bar{p}_t = (1-\alpha)\bar{p}_{t-1} + \alpha p_t.$$

    Our policy is then given by

    $$X^\pi(S_t|\theta) = \begin{cases} 1 & \text{If } p_t \leq \bar{p}_t - \theta \\ 0 & \text{Otherwise.} \end{cases}$$

    The function $X^\pi(S_t|\theta)$ decreases monotonically in p_t (as p_t increases, $X^\pi(S_t|\theta)$ goes from 1 to 0).
  - A shuttle bus waits until there are at least R_t customers on the bus, or it has waited τ_t. The decision to dispatch goes from $x_t = 0$ (hold the bus) to $x_t = 1$ (dispatch the bus) as R_t exceeds a threshold θ^R or as τ_t exceeds θ^τ, which means the policy $X^\pi(S_t|\theta)$ increases monotonically in both state variables $S_t = (R_t, \tau_t)$.
- A battery is being used to buy power from the grid when electricity prices p_t fall below a lower limit θ^{min}, or sell when the price goes above θ^{max}. The battery does nothing when $\theta^{min} < p_t < \theta^{max}$. We write the policy as

  $$X^\pi(S_t|\theta) = \begin{cases} \text{-1} & \text{If } p_t \leq \theta^{min} \\ 0 & \text{If } \theta^{min} < p_t < \theta^{max} \\ 1 & \text{If } p_t \geq \theta^{max} \end{cases}$$

  We see that $X^\pi(S_t|\theta)$ increases monotonically in the state $S_t = p_t$.
- The rate at which water should be released from a reservoir will increase with the reservoir level. The rate is specified as a parameterized lookup table, with a different rate for each range of reservoir levels.
- Dosages for blood sugar control increase with both the weight of the patient, and with the patient's glycemic index. The policy is in the form of a lookup table, with different dosages for each range of weight and glycemic index.

Each of these policies is controlled by a relatively small number of parameters, although this is not always the case. For example, if we use a fine discretization of the patient's weight and glycemic index, we could find that we need to specify hundreds of dosages. However, monotonicity can dramatically reduce the search region.

12.1.6 Nonparametric policies

The problem with parametric models is that sometimes functions are simply too complex to fit with low-order parametric models. For example, imagine that our policy looks like the function shown in figure 12.1. Simple quadratic fits will not work, and higher-order

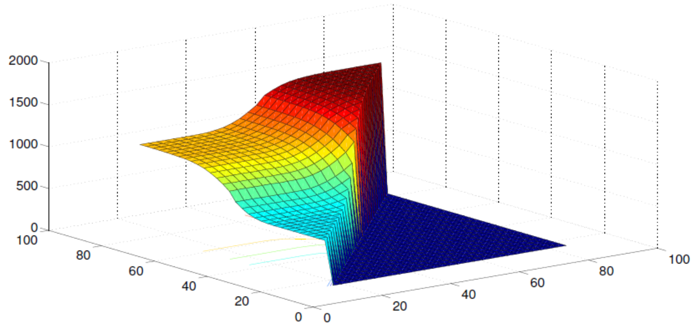


Figure 12.1 Illustration of a complex nonlinear function.

polynomials will struggle due to overfitting unless the number of observations is extremely large.

We could handle very general functions if we could use lookup tables (which may require that we discretize any continuous parameters). However, lookup tables can become extremely large when we have three or more dimensions in our state variable. Even three dimensional lookup tables quickly grow to thousands to millions of elements. The problem is compounded when the search algorithm has to evaluate actions for each state many times to handle noise.

Nonparametric models, simply stated, use local averaging to help smooth out a surface. Imagine we have an unknown function $f(s)$ (our policy) for each state s (assume that s has between 1 and 3 dimensions). Now imagine that we can evaluate (with noise) the function at a set of states $s^1, \dots s^n$ (imagine, for example, that we are sampling 10 percent of the states), and let

$$\hat{f}^k = f(s^k) + \varepsilon^k$$

be our noisy observation of the function. We can construct an approximation $\bar{f}(s)$ of the function for each state s (even if we never observe s) using

$$\bar{f}^n(s) = \sum_{m=1}^{n} \frac{k(s^m, s)\hat{f}^n}{\sum_{m'=1}^{n} k(s^{m'}, s)}. \tag{12.3}$$

Here, the function $k(s^m, s)$ is a *kernel function* which is a measure of the distance between s^m and s. A popular kernel function is

$$k(s^m, s) = \exp - \frac{|s^m - s|}{\beta},$$

where β is a parameter known as the “bandwidth” which performs scaling of the state variable.

There are a number of functional forms for the kernel. Another strategy is to simply take a set of nearest neighbors and average them. Regardless of the weighting procedure, the basic idea is to represent a function using a weighted average of nearby points. An advantage of nonparametric models is that they make it possible to approximate functions with a relatively small number of observations, without imposing any structure on the function. However, this is also a disadvantage, because it means you are not allowed to impose any structure, and you may get odd behaviors due to noise.

It is hard to search over a space of nonparametric policies, since there is no real functional form to search. Instead, an approach that has been used in robotics is local parametric approximations, where the policy is represented by a parametric model defined over specific regions. These regions are typically specified by a domain expert who understands the different behaviors that might arise in different regions.

12.1.7 Neural network policies

Neural networks have emerged as a powerful approximation architecture in machine learning where they have received considerable attention for their use in pattern recognization settings. Neural networks have actually been used for decades in primarily deterministic engineering control problems. We first introduced readers to neural networks in section 3.9.3, laying the foundation for their use in the entire spectrum of approximation challenges that arise in stochastic optimization.

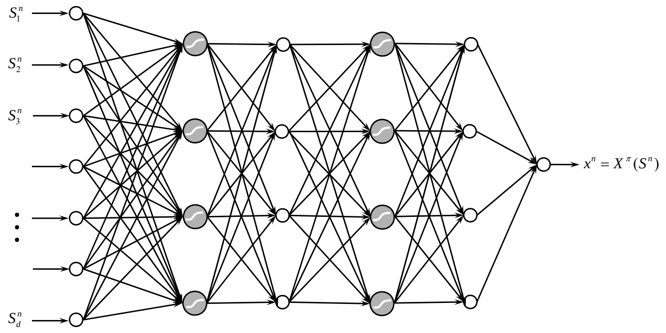


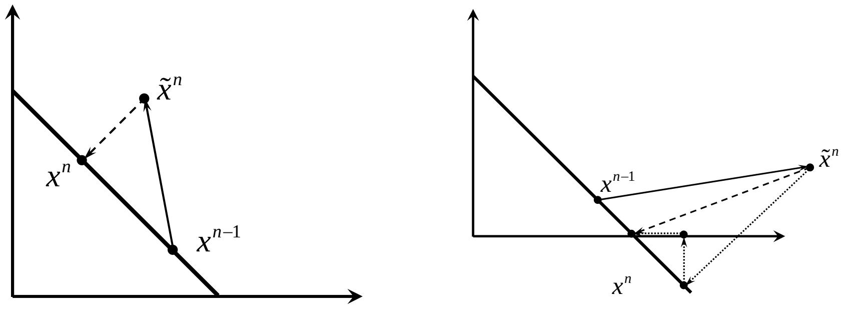
Figure 12.2 Illustration of a four-layer neural network.

Figure 12.2 illustrates a four layer neural network, mapping a d-dimensional state vector S^n to an output $x^n = X^\pi(S^n|\theta)$ which represents an approximate policy mapping state to a decision, where θ represents the weights connecting the nodes within the network (typically θ has dozens to hundreds of dimensions in smaller networks, and thousands of dimensions in deep networks. In most engineering applications, the decision $x = X^\pi(S)$ is typically a low-dimensional vector of continuous controls, but neural networks have been used for discrete classification problems.

12.1.8 Constraints

An issue that arises with policy function approximations is the handling of constraints, since it can be difficult or impossible to design analytical functions that guarantee that a decision satisfies a set of constraints. Constraints are typically handled using a simple projection. This is represented mathematically by a projection operator $\Pi_{\mathcal{X}}(x)$ (nothing to do with policies) that maps a point x onto a region $\mathcal{X}$. So, we would write our policy using

$$x_t = \Pi_{\mathcal{X}_t}[X_t^\pi(S_t)].$$



(a): Simple projection (b): Compound projection

Figure 12.3 Illustration of projection onto a linear feasible region.

The easiest constraints to handle are box constraints of the form $0 \leq x_t \leq u_t$ where u_t are upper bounds on each dimension of x_t. In this case, if our function $X_t^\pi(S_t)$ returns a (vector-valued) decision x_t, we simply have to check each dimension of x_t and impose these constraints (elements less than 0 are set equal to 0, while elements greater than their corresponding value in u_t are set to the value in u_t).

Slightly harder are constraints of the form $Ax = b_t$ or $Ax \leq b_t$. The project process is illustrated in figure 12.3. Figure 12.3a demonstrates a basic projection of a point $\tilde{x}^n$ from outside of the feasible region back onto the feasible region. This is done as an *orthogonal projection*. Imagine that our plane is at a 45 degree angle (as would arise if our constraint was of the form $x_1 + x_2 \leq 10$). Then, we have to adjust $\tilde{x}^n$ by subtracting equal quantities from each dimension until we are back on the plane. For example, if $\tilde{x}_1 = 6$ and $\tilde{x}_2 = 8$ so that $\tilde{x}_1 + \tilde{x}_2 = 14$, then we would take how far we are in violation (14-10 = 4), divide by the two dimensions (getting 2) and then subtract 2 from each dimension.

It could easily be the case that this projection simply creates a new violation in the form of one of the variables becoming negative, as illustrated in figure 12.3b. In this case, we simply zero out the negative dimension, then eliminate this dimension from further calculations (simply set it to zero) and repeat the process.

For more general problems, we have to fall back on the formal definition of the projection operator, which involves minimizing the distance between the point x and the feasible region $\mathcal{X}$. The most standard definition is

$$\Pi_{\mathcal{X}}[x] = \underset{x' \in \mathcal{X}}{\arg\min} \, \|x - x'\|_2, \tag{12.4}$$

where $\|x - x'\|_2$ is the "L_2 norm" defined by

$$\|x - x'\|_2 = \sum_i (x_i - x_i')^2.$$

The complexity of solving the nonlinear programming problem in (12.4) depends on the nature of the feasible region $\mathcal{X}$.

12.2 POLICY SEARCH

Given a parametric (or locally parametric) function parameterized by θ (typically a vector, but not always), we now face the challenge of finding the best value of θ. There are different styles of policy search:

Derivative-based vs. derivative free In some cases we can approximate derivatives with respect to θ, although these are typically quite approximate. Alternatively we can use the derivative-free methods in chapter 7, although it is likely that this will be limited to low-dimensional parameter vectors.

Online vs. offline learning In online learning, we are learning in an environment where updates come to us. As a rule, we have to live with the performance of our policy, which means we are maximizing the cumulative reward. Most policy search uses some form of adaptive algorithm, although this can be done in a laboratory where we use one policy, the *learning policy* to find the best policy to implement, called the *implementation policy*.

Stationary vs. nonstationary environments Most of the analysis of algorithms is performed in the context of stationary (possibly even static) environments, where exogenous information comes from a single distribution. When working in online settings (in the field), it is more often the case that data is coming from a nonstationary setting.

Performance-based vs. supervised learning Most policy search uses as a goal to maximize the total reward (either the terminal reward or cumulative reward), but there are settings where we have an "expert" (the supervisor) who will specify what to do, allowing us to fit our policies to the choices of the supervisor.

Policy search is part of a long history of optimizing simulations. The simplest approach involved testing M different designs $x_1, \ldots, x_M$ and picking the one that was best. This is the problem class we considered in chapter 7 for derivative-free optimization.

Now consider what happens when x is a continuous vector (or even a vector of integers). A popular problem involved the design of queueing networks that might arise in a manufacturing setting. A job might move from machine i to machine j, where it might have to sit in a buffer waiting to be worked on. If the buffer is full, the job has to go to a different machine. The question is to determine how large the buffers should be. Let $x_1, \ldots, x_K$ be the size of each of the buffers. We now have the problem of searching a K-dimensional vector.

One approach for optimizing designs (which we return to) is to use finite-difference methods that we introduced in section 5.4.3. This can get expensive when K is large (even 10 dimensions can be quite expensive), since we need at least $K+1$ simulations to estimate a gradient. In the 1980's, researchers developed methods for estimating gradients from a single run of a simulator using a technique called *infinitesimal perturbation analysis*, or IPA. IPA techniques were specifically designed for queueing networks, but highlighted the power of being able to estimate gradients from a single run of a simulator.

Policy search shares a common starting point with simulation optimization. Instead of choosing a design x, we have to find the parameters θ that govern a policy. Both are design parameters that have to be chosen before we run our simulator. However, the process of finding derivatives with respect to the policy control parameters θ is quite different than finding the derivatives with respect to the size of buffers in a queueing network.

We divide our discussion primarily along the two fundamental search strategies: derivative-based and derivative-free. Derivative-based methods are attractive because they allow us to draw on the foundation we provided in chapter 5, which is the only practical way (at this time) to handle high dimensional vectors of parameters, as might arise when our policy is represented by a neural network. Derivative-based policy search starts from writing the value of a policy as

$$F^\pi(\theta) = \mathbb{E}\left\{\sum_{t=0}^{T} C(S_t, X^\pi(S_t|\theta))|S_0\right\}, \tag{12.5}$$

where $S_{t+1} = S^M(S_t, X^\pi(S_t|\theta), W_{t+1})$. If we let $W = (W_1, \ldots, W_T)$, then this is precisely

$$\max_\theta \mathbb{E}F(\theta, W), \tag{12.6}$$

where we dropped the "π" superscript because in this setting, the structure of the policy has been fixed and is otherwise determined by θ. This is now the same problem we faced in chapter 5, where we can search for θ using a standard stochastic gradient algorithm

$$\theta^{n+1} = \theta^n + \alpha_n \nabla_\theta F^\pi(\theta^n, W^{n+1}). \tag{12.7}$$

The challenge, then, is finding the gradient $\nabla_\theta F(\theta^n, W^{n+1})$, a problem we did not address in chapter 5. To do this, it is useful to identify three classes of problems:

Discrete dynamic programs - These are problems where we are at a node (state) s, choose a discrete action a and then transition to a node s' with probability $P(s'|s,a)$ (which we represent but generally cannot compute). An important subclass of graph problems are those where actions are chosen at random (known as a stochastic policy), but transitions are made deterministically. Here, we wish to optimize a parameterized policy $A^\pi(s|\theta)$, where action $a_t = A^\pi(S_t|\theta)$ is discrete.

Control problems - In this setting we choose a continuous control u_t that impacts the state S_{t+1} in a continuous way through a known (and differentiable) transition function.

Resource allocation - Here, we have a vector of resources R_t which we move with a vector x_t to produce a new allocation R_{t+1}, possibly with random perturbations, according to the equation

$$R_{t+1} = R_t + A_t x_t + \hat{R}_{t+1},$$

where R_t and x_t are vectors, and A_t is a suitably defined matrix. This problem class includes all of the physical resource allocation problems described in chapter 8. For this problem class, we wish to optimize a parameterized policy $X^\pi(s|\theta)$, where $x_t = X^\pi(S_t|\theta)$ is typically a vector (possibly high dimensional).

All three problem classes have attracted considerable attention, and illustrate different methods for computing gradients. We begin with section 12.3 which describes a powerful result known as the policy gradient theorem for taking derivatives of policies for discrete dynamic programs, where actions are discrete (in particular, they are categorical). We then transition to control problems where we can exploit the derivative of a downstream state with respect to a control.

We defer the discussion of resource allocation problems to chapter 13 where they are presented in the context of parameterized cost function approximations, which represent a popular way of handling this problem class. The complication here is that the policy has an imbedded minimization problem.

12.3 THE POLICY GRADIENT THEOREM FOR DISCRETE DYNAMIC PROGRAMS

We assume that we are going to maximize the single-period expected reward in steady state. We use the following notation

$r(s,a)$ = Reward if we are in state $s \in \mathcal{S}$ and take action $a \in \mathcal{A}_s$,

$A^\pi(s|\theta)$ = Policy that determines the action a given that we are in state s, which is parameterized by θ,

$P_t(s'|s,a)$ = Probability of transitioning to state s' given that we are in state s and take action a at time t (we use $P(s'|s,a)$ if the underlying dynamics are stationary),

$d^\pi_t(s|\theta)$ = Probability of being in state s at time t while following policy π,

We first introduce a parameterized stochastic policy which is typically required for problems where decisions are discrete with no particular structure (e.g. red-green-blue). We note that the parameters that we are optimizing are primarily controlling the balance of exploration and exploitation. We then present the objective function (there is more than one way to write this, as we show later). Finally, we describe a computable method for taking the gradient of this objective function.

12.3.1 A stochastic policy

We follow the standard practice in the literature of using what is called a stochastic policy, where an action a is chosen probabilistically. We represent our policy using

$p^\pi_t(a|s,\theta)$ = The probability of choosing action a at time t, given that we are in state s, where θ is a tunable parameter (possibly a vector).

Most of the time we will use a stationary policy that we denote $\bar{p}^\pi(a|s,\theta)$ which can be viewed as a time-averaged version of our policy $p^\pi_t(a|s,\theta)$ which we might compute using

$$\bar{p}^\pi(a|s,\theta) = \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} p^\pi_t(a|s,\theta).$$

A particularly popular policy (especially in computer science) assumes that actions are chosen at random according to a Boltzmann distribution (also known as Gibbs sampling). Assume at time t that we have

$\bar{Q}_t(s,a)$ = Estimated value at time t of being in state s and taking action a minus the steady state .

Now define the probabilities (using our familiar Boltzmann distribution)

$$p^\pi_t(a|s,\theta) = \frac{e^{\theta \bar{Q}_t(s,a)}}{\sum_{a'\in\mathcal{A}_s} e^{\theta \bar{Q}_t(s,a')}}. \tag{12.8}$$

We can compute the values $\bar{Q}_t(s,a)$ using $\bar{Q}_t(s,a) = r(s,a)$, although this means choosing actions based on immediate rewards. Alternatively, we might use

$$\bar{Q}_t(s,a) \quad = \quad r(s,a) + \max_{a'} \bar{Q}_{t+1}(s',a'),$$

where s' is chosen randomly from simulating the next step (or sampling from the transition matrix $P_t(s'|s,a)$ if this is available). We first saw methods for computing Q-values under the umbrella of reinforcement learning in section 2.1.10.

If we are modeling a stationary problem, it is natural to transition to a stationary policy. Let $\bar{p}^\pi(a|s,\theta)$ be our stationary action probabilities where we replace the time-dependent values $\bar{Q}_t(s,a)$ with stationary values $\bar{Q}(s,a)$ computed using

$$\bar{Q}^\pi(s,a|\theta) \quad = \quad r(s,a) + \mathbb{E}\left\{\sum_{t'=1}^{T} r(S_{t'}, A^\pi(S_{t'}|\theta))|S_0 = s, a_0 = a\right\}. \quad (12.9)$$

This is the total reward over the horizon from starting in state s and taking action a (note that we could use average or discounted rewards, over finite or infinite horizons). We remind the reader we are never going to actually compute these expectations. Using these values, we can create a stationary distribution for choosing actions using

$$\bar{p}^\pi(a|s,\theta) = \frac{e^{\theta\bar{Q}^\pi(s,a|\theta)}}{\sum_{a'\in\mathcal{A}_s} e^{\theta\bar{Q}^\pi(s,a'|\theta)}}, \quad (12.10)$$

Finally, our policy $A^\pi(s|\theta)$ is to choose action a with probability given by $p_t^\pi(a|s,\theta)$. The development below does not require that we use the Boltzmann policy, but it helps to have an example in mind.

12.3.2 The objective function

To develop the gradient, we have to start by writing out our objective function which is to maximize the average reward over time, given by

$$F^\pi(\theta) = \lim_{T\to\infty}\frac{1}{T}\left\{\sum_{t=0}^{T}\sum_{s\in\mathcal{S}}\left(d_t^\pi(s|\theta)\sum_{a\in\mathcal{A}_s} r(s,a)p_t^\pi(a|s,\theta)\right)\right\}. \quad (12.11)$$

A more compact form involves replacing the time-dependent state probabilities with their time averages (since we are taking the limit). Let

$$\bar{d}^\pi(s|\theta) = \lim_{t\to\infty}\frac{1}{t}d_t^\pi(s|\theta).$$

We can then write our average reward per time period as

$$F^\pi(\theta) = \sum_{s\in\mathcal{S}}\bar{d}^\pi(s|\theta)\sum_{a\in\mathcal{A}_s} r(s,a)\bar{p}^\pi(a|s,\theta). \quad (12.12)$$

12.3.3 The policy gradient

We are now ready to take derivatives. Differentiating both sides of (12.12) gives us

$$\nabla_\theta F^\pi(\theta) \quad = \quad \sum_{s\in\mathcal{S}}\left(\nabla_\theta\bar{d}^\pi(s|\theta)\sum_{a\in\mathcal{A}_s} r(s,a)\bar{p}^\pi(a|s,\theta) + \bar{d}^\pi(s|\theta)\sum_{a\in\mathcal{A}_s} r(s,a)\nabla_\theta\bar{p}^\pi(a|s,\theta)\right). \quad (12.13)$$

While we cannot compute probabilities such as $d^\pi(s)$, we can simulate them (we show this below). We also assume we can compute $\nabla_\theta\bar{p}^\pi(a|s,\theta)$ by differentiating our probability distribution in (12.10). Derivatives of probabilities such as $\nabla_\theta \bar{d}^\pi(s|\theta)$, however, are another matter.

This is where the development known as the *policy gradient theorem* helps us. This theorem tells us that we can calculate the gradient of $F^\pi(\theta)$ with respect to θ using

$$\frac{\partial F^\pi(\theta)}{\partial \theta} = \sum_s d^\pi(s|\theta) \sum_a \frac{\partial \bar{p}^\pi(a|s,\theta)}{\partial \theta} Q^\pi(s,a). \tag{12.14}$$

where $Q^\pi(s,a)$ (defined below) is the expected difference between rewards earned each time period from a starting state, and the expected reward (given by $F^\pi(\theta)$) earned each period when we are in steady state. We will not be able to compute this derivative exactly, but we show below that we can produce an unbiased estimate without too much difficulty. What is most important is that, unlike equation (12.13), we do not have to compute (or even approximate) $\nabla_\theta \bar{d}^\pi(s|\theta)$.

We are going to begin by defining two important quantities:

$$\begin{aligned} Q^\pi(s,a|\theta) &= \sum_{t=1}^{\infty} \mathbb{E}\{r(s_t,a_t) - F^\pi(\theta)|s_0 = s, a_0 = a\}, \\ V^\pi(s|\theta) &= \sum_{t=1}^{\infty} \mathbb{E}\{r(s_t,a_t) - F^\pi(\theta)|s_0 = s\}, \\ &= \sum_{a\in\mathcal{A}} \bar{p}^\pi(a_0 = a|s,\theta) \sum_{t=1}^{\infty} \mathbb{E}\{r(s_t,a_t) - F^\pi(\theta)|s_0 = s, a_0 = a\}, \\ &= \sum_a \bar{p}^\pi(a|s,\theta) Q^\pi(s,a). \end{aligned} \tag{12.15}$$

Note that $Q^\pi(s,a|\theta)$ is quite different than the quantities $\bar{Q}^\pi(s,a|\theta)$ used above for the Boltzmann policy (which is consistent with Q-learning, which we first saw in section 2.1.10). $Q^\pi(s,a|\theta)$ sums the difference between the reward each period and the steady state reward per period (a difference that goes to zero on average), given that we start in state s and initially take action a. $V^\pi(s|\theta)$ is simply the expectation over all initial actions actions a as specified by our probabilistic policy

We next rewrite $Q^\pi(s,a)$ as the first term in the summation, plus the expected value of the remainder of the infinite sum using

$$\begin{aligned} Q^\pi(s,a) &= \sum_{t=1}^{\infty} \mathbb{E}\{r_t - F^\pi(\theta)|s_0 = s, a_0 = a\}, \\ &= r(s,a) - F^\pi(\theta) + \sum_{s'} P(s'|s,a) V^\pi(s'), \quad \forall s,\ a, \end{aligned} \tag{12.16}$$

where $P(s'|s,a)$ is the one-step transition matrix (recall that this does not depend on θ). Solving for $F^\pi(\theta)$ gives

$$F^\pi(\theta) = r(s,a) + \sum_{s'} P(s'|s,a) V^\pi(s') - Q^\pi(s,a). \tag{12.17}$$

Now, note that $F^\pi(\theta)$ is not a function of either s or a, even though they both appear in the right hand side of (12.17). Noting that since our policy must pick some action,

$\sum_{a\in\mathcal{A}} \bar{p}^\pi(a|s,\theta) = 1$, which means

$$\sum_{a\in\mathcal{A}} \bar{p}^\pi(a|s,\theta)F^\pi(\theta) = F^\pi(\theta), \quad \forall a.$$

This means we can take the expectation of (12.17) over all actions, giving us

$$F^\pi(\theta) = \sum_a \bar{p}^\pi(a|s,\theta)\left(r(s,a) + \sum_{s'} P(s'|s,a)V^\pi(s') - Q^\pi(s,a)\right), \tag{12.18}$$

for all states s. We can now take derivatives using the following steps (explanations follow the equations):

$$\begin{aligned}
\frac{\partial F^\pi(\theta)}{\partial\theta} &= \frac{\partial}{\partial\theta}\left(\sum_a \bar{p}^\pi(a|s,\theta)\Big(r(s,a) + \sum_{s'} P(s'|s,a)V^\pi(s') - Q^\pi(s,a)\Big)\right) && (12.19)\\
&= \sum_a \frac{\partial\bar{p}^\pi(a|s,\theta)}{\partial\theta} r(s,a) + \sum_a \frac{\partial\bar{p}^\pi(a|s,\theta)}{\partial\theta}\sum_{s'} P(s'|s,a)V^\pi(s')\\
&\quad + \sum_a \bar{p}^\pi(a|s,\theta)\sum_{s'} P(s'|s,a)\frac{\partial V^\pi(s')}{\partial\theta} - \frac{\partial}{\partial\theta}\Big(\sum_a \bar{p}^\pi(a|s,\theta)Q^\pi(s,a)\Big) && (12.20)\\
&= \sum_a \frac{\partial\bar{p}^\pi(a|s,\theta)}{\partial\theta}\Big(r(s,a) + \sum_{s'} P(s'|s,a)V^\pi(s')\Big)\\
&\quad + \sum_a \bar{p}^\pi(a|s,\theta)\sum_{s'} P(s'|s,a)\frac{\partial V^\pi(s')}{\partial\theta} - \frac{\partial V^\pi(s)}{\partial\theta} && (12.21)\\
&= \sum_a \frac{\partial\bar{p}^\pi(a|s,\theta)}{\partial\theta}\Big(Q^\pi(s,a) + F^\pi(\theta)\Big)\\
&\quad + \sum_a \bar{p}^\pi(a|s,\theta)\sum_{s'} P(s'|s,a)\frac{\partial V^\pi(s')}{\partial\theta} - \frac{\partial V^\pi(s)}{\partial\theta} && (12.22)\\
&= \sum_a \frac{\partial\bar{p}^\pi(a|s,\theta)}{\partial\theta} Q^\pi(s,a) + \sum_a \bar{p}^\pi(a|s,\theta)\sum_{s'} P(s'|s,a)\frac{\partial V^\pi(s')}{\partial\theta} - \frac{\partial V^\pi(s)}{\partial\theta}. && (12.23)
\end{aligned}$$

Equation (12.19) is from (12.18); (12.20) is the direct expansion of (12.19), where two terms vanish because $r(s,a)$ and $P(s'|s,a)$ do not depend on the policy $\bar{p}^\pi(a|s,\theta)$; (12.19) uses (12.15) for the last term; (12.22) uses (12.16); (12.15) uses the fact $F^\pi(\theta)$ is constant over states and actions, and $\sum_a \bar{p}^\pi(a|s,\theta) = 1$. Finally, note that equation (12.23) is true for all states.

We proceed to write

$$\begin{aligned}
\frac{\partial F^\pi(\theta)}{\partial\theta} &= \sum_s d^\pi(s|\theta)\frac{\partial F^\pi(\theta)}{\partial\theta} && (12.24)\\
&= \sum_s d^\pi(s|\theta)\left(\sum_a \frac{\partial\bar{p}^\pi(a|s,\theta)}{\partial\theta} Q^\pi(s,a)\right.\\
&\quad \left. + \sum_a \bar{p}^\pi(a|s,\theta)\sum_{s'} P(s'|s,a)\frac{\partial V^\pi(s')}{\partial\theta} - \frac{\partial V^\pi(s)}{\partial\theta}\right). && (12.25)
\end{aligned}$$

Expanding gives us

$$\begin{aligned}
\frac{\partial F^\pi(\theta)}{\partial\theta} =& \sum_s d^\pi(s|\theta)\sum_a \frac{\partial\bar{p}^\pi(a|s,\theta)}{\partial\theta}Q^\pi(s,a) \\
&+\sum_s d^\pi(s|\theta)\sum_a \bar{p}^\pi(a|s,\theta)\sum_{s'} P(s'|s,a)\frac{\partial V^\pi(s')}{\partial\theta} - \sum_s d^\pi(s|\theta)\frac{\partial V^\pi(s)}{\partial\theta} \qquad (12.26)\\
=& \sum_s d^\pi(s|\theta)\sum_a \frac{\partial\bar{p}^\pi(a|s,\theta)}{\partial\theta}Q^\pi(s,a) \\
&+\sum_s d^\pi(s|\theta)\frac{\partial V^\pi(s)}{\partial\theta} - \sum_s d^\pi(s|\theta)\frac{\partial V^\pi(s)}{\partial\theta} \qquad (12.27)\\
=& \sum_s d^\pi(s|\theta)\sum_a \frac{\partial\bar{p}^\pi(a|s,\theta)}{\partial\theta}Q^\pi(s,a). \qquad (12.28)
\end{aligned}$$

Equation (12.24) uses $\sum_s d^\pi(s|\theta) = 1$; (12.25) uses the fact (12.23) holds for all s; (12.26) simply expands (12.25); (12.27) uses the property that since $d^\pi(s)$ is the stationary distribution, then $\sum_s d^\pi(s|\theta)P(s'|s,a) = d^\pi(s'|\theta)$ (after substituting this result, then just change the index from s' to s). Equation (12.28) is the policy gradient theorem we first presented in equation (12.14).

12.3.4 Computing the policy gradient

As is always the case in stochastic optimization, the challenge boils down to computation. To help the discussion, we repeat the policy gradient result:

$$\frac{\partial F^\pi(\theta)}{\partial\theta} = \sum_s d^\pi(s|\theta)\sum_a \frac{\partial\bar{p}^\pi(a|s,\theta)}{\partial\theta}Q^\pi(s,a). \qquad (12.29)$$

We start by assuming that we have some analytical form for the policy which allows us to compute $\partial\bar{p}^\pi(a|s,\theta/\partial\theta$ (which is the case when we use our Boltzmann distribution). This leaves the stationary probability distribution $d^\pi(s|\theta)$, and the marginal rewards $Q^\pi(s,a)$.

Instead of computing $d^\pi(s|\theta)$ directly, we instead simply simulate the policy, depending on the fact that over a long simulation, we will visit each state with probability $d^\pi(s|\theta)$. Thus, for large enough T, we can compute

$$\nabla_\theta F^\pi(\theta) \approx \frac{1}{T}\sum_{t=1}^T\sum_a \frac{\partial\bar{p}^\pi(a|s_t,\theta)}{\partial\theta}Q^\pi(s_t,a), \qquad (12.30)$$

where we simulate according to a known transition function $s_{t+1} = S^M(s_t,a,W_{t+1})$. We may simulate the process from a known transition function and a model of the exogenous information process W_t (if this is present), or we may simply observe the policy in action over a period of time.

This then leaves us with $Q^\pi(s_t,a)$. We are going to approximate this with estimates that we call $\bar{Q}^\pi_t(S_t|\theta)$, which we will compute by running a simulation starting at time t until T (or some horizon $t+H$). This requires running a different simulation that can be called a roll-out simulation, or a lookahead simulation. To avoid confusion, we are going to let $\tilde{S}_{tt'}$ be the state variable at time t' in a roll-out simulation that is initiated at time t. We let $W_{tt'}$ be the simulated random information between $t'-1$ and t' for a simulation that is initiated at time t. Recognizing that $\tilde{S}_{tt} = S_t$, we can write

$$\bar{Q}^\pi_t(S_t|\theta) = \mathbb{E}_W \frac{1}{T-t} \sum_{t'=t}^{T-1} r(\tilde{S}_{tt'}, A^\pi(\tilde{S}_{tt'}|\theta)),$$

where $\tilde{S}_{t,t'+1} = S^M(\tilde{S}_{tt'}, A^\pi(\tilde{S}_{tt'}|\theta), \widetilde{W}_{t,t'+1})$ represents the transitions in our lookahead simulation. Of course, we cannot compute the expectation, so instead we use the simulated estimate

$$\bar{Q}^\pi_t(S_t|\theta) \approx \frac{1}{T-t} \sum_{t'=t}^{T-1} r(\tilde{S}_{tt'}, A^\pi(\tilde{S}_{tt'}|\theta)). \qquad (12.31)$$

We note that while we write this lookahead simulation as spanning the period from t to T, this is not necessary. We might run these lookahead simulations over a fixed interval $(t, t+H)$, and adjust the averaging accordingly.

We now have a computable estimate of $F^\pi(\theta)$ which we obtain from (12.31) by replacing $Q^\pi_t(S_t|\theta)$ with $\bar{Q}^\pi_t(S_t|\theta)$, giving us a sampled estimate of policy π using

$$F^\pi(\theta) \quad \approx \quad \sum_{t=0}^{T-1} \hat{Q}^\pi_t(S_t|\theta).$$

The final step is actually computing the derivative $\nabla_\theta F^\pi(\theta)$. For this, we are going to turn to numerical derivatives. Assume the lookahead simulations are fairly easy to compute. We can then obtain estimates of $\nabla_\theta \hat{Q}^\pi_t(S_t|\theta)$ using the finite difference. We can do this by perturbing each element of θ. If θ is a scalar, we might use

$$\nabla_\theta \hat{Q}^\pi_t(S_t|\theta) = \frac{\hat{Q}^\pi_t(S_t|\theta+\delta) - \hat{Q}^\pi_t(S_t|\theta-\delta)}{2\delta} \qquad (12.32)$$

If θ is a vector, we might do finite differences for each dimension, or turn to simultaneous perturbation stochastic approximation (SPSA) (see section 5.4.3 for more details).

This strategy was first introduced under the name of the REINFORCE algorithm. It has the nice advantage of capturing the downstream impact of changing θ on later states, but in a very brute force manner.

12.4 DERIVATIVE-BASED POLICY SEARCH: CONTROL PROBLEMS

In this section, we are going to assume that we are trying to find a control policy $U^\pi(S_t|\theta)$ (known as a control law in the engineering community) parameterized by θ that returns a continuous, vector-valued control $u_t = U^\pi(S_t|\theta)$. Using our control notation, our optimization problem would be written

$$F^\pi(\theta) = \mathbb{E}\left\{\sum_{t=0}^{T} C(S_t, U^\pi_t(S_t|\theta))|S_0\right\}, \qquad (12.33)$$

where our dynamics evolve (as before) according to

$$S_{t+1} = S^M(S_t, u_t, W_{t+1}),$$

where we are given an initial state S_0 and access to observations of the sequence $W = (W_1, \ldots, W_T)$. We have written our policy $U_t^\pi(S_t)$ in a time-dependent form for generality, but this means estimating time-dependent parameters θ_t that characterize the policy. In most applications we would use the stationary version $U^\pi(S_t)$, with a single set of parameters θ.

Control problems are distinguished from discrete dynamic programs specifically because we assume we can compute $\partial S_{t+1}/\partial u_t$. With discrete dynamic programs, we assumed the actions a were categorical (e.g. left/right or red/green/blue). In that setting, we had to consider the downstream impact of a decision made now by capturing the effect of changing the policy parameter θ on the probability of which state we would visit. Now we can capture this impact directly.

There are two approaches for minimizing $F^\pi(\theta)$ over the parameter vector θ:

Batch learning Here we replace (12.33) with an average over N samples, giving us

$$\overline{F}^\pi(\theta) = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{T} C(S_t(\omega^n), U_t^\pi(S_t(\omega^n)|\theta)), \tag{12.34}$$

where $S_{t+1}(\omega^n) = S^M(S_t(\omega^n), U^\pi(S_t(\omega^n)), W_{t+1}(\omega^n))$ is the sequence of states generated following sample path ω^n. This is a classical statistical estimation problem.

Adaptive learning Rather than solving a single (possibly very large) batch problem, we can use our standard stochastic gradient updating logic (from chapter 7)

$$\theta^{n+1} = \theta^n + \alpha_n \nabla_\theta F^\pi(\theta^n, W^{n+1}).$$

This update is executed following each forward pass through the simulation.

Both approaches depend on computing the gradient $\nabla_\theta F^\pi(\theta, W)$ for a given simple path ω from which we generate a sequence of state $S_{t+1} = S^M(S_t, u_t, W_{t+1}(\omega))$ where $u_t = U^\pi(S_t)$. Normally we would write $S_t(\omega)$ or $u_t(\omega)$ to indicate the dependence on sample path ω, but we suppress this here for notational compactness.

We find the gradient by differentiating (12.33) with respect to θ, which requires a meticulous application of the chain rule, recognizing that the contribution $C(S_t, u_t)$ is a function of both S_t and u_t, the policy $U^\pi(S_t|\theta)$ is a function of both the state S_t and the parameter θ, and the state S_t is a function of the previous state S_{t-1}, the previous control u_{t-1}, and the most recent exogenous information W_t (which is assumed to be independent of the control, although this could be handled). This gives us

$$\begin{aligned}\nabla_\theta F^\pi(\theta,\omega) \;=\; & \left(\frac{\partial C_0(S_0,u_0)}{\partial u_0}\right)\left(\frac{\partial U_0^\pi(S_0|\theta)}{\partial \theta}\right) + \sum_{t'=1}^{T}\Bigg[\left(\frac{\partial C_{t'}(S_{t'},U_{t'}^\pi(S_{t'}))}{\partial S_{t'}}\frac{\partial S_{t'}}{\partial \theta}\right) \\ & + \frac{\partial C_{t'}(S_{t'},u_{t'})}{\partial u_{t'}}\left(\frac{\partial U_{t'}^\pi(S_{t'}|\theta)}{\partial S_{t'}}\frac{\partial S_{t'}}{\partial \theta} + \frac{\partial U_{t'}^\pi(S_{t'}|\theta)}{\partial \theta}\right)\Bigg]\end{aligned} \tag{12.35}$$

where

$$\frac{\partial S_{t'}}{\partial \theta} = \frac{\partial S_{t'}}{\partial S_{t'-1}}\frac{\partial S_{t'-1}}{\partial \theta} + \frac{\partial S_{t'}}{\partial u_{t'-1}}\left[\frac{\partial U_{t'-1}^\pi(S_{t'-1}|\theta)}{\partial S_{t'-1}}\frac{\partial S_{t'-1}}{\partial \theta} + \frac{\partial U_{t'-1}^\pi(S_{t'-1})}{\partial \theta}\right] \tag{12.36}$$

The derivatives $\partial S_{t'}/\partial\theta$ are computed using (12.36) by starting at $t' = 0$ where

$$\frac{\partial S_0}{\partial \theta} = 0,$$

and stepping forward in time.

Equations (12.35) and (12.36) require that we be able to take derivatives of the cost function, the policy, and the transition function. We assume this is possible, although the complexity of these derivatives is highly problem dependent.

12.5 NEURAL NETWORK POLICIES

A popular strategy (especially in engineering) for representing policies is to use neural networks, which are well suited for low-dimensional continuous control problems (which are typically deterministic). We first introduced neural networks in chapter 3, both as a form of parametric model (when there is a small number of layers), or as a nonparametric model in the setting where we use a large (effectively infinite) number of layers (often known as deep learning).

Neural networks are able to model very complex functions, although their richness makes them vulnerable to overfitting which can only be overcome through considerable training. They are well suited to handling high-dimensional state-spaces, although they work better with lower-dimensional control spaces (most engineering applications involve no more than 10 or 20 dimensions).

Our presentation of policy builds on the foundation provided in section 12.1.7. We present our ideas using the setting of a three-layer architecture. The input layer captures the d-dimensional state variable, S_t, the hidden layer aims at discovering nonlinear feature transformations of the state, while the output represents the m-dimensional real-valued control $\boldsymbol{u}_t$. The first layer activations are set to:

$$\boldsymbol{a}^{(1)} = \begin{bmatrix} 1 \\ a_1^{(1)} \\ \vdots \\ a_d^{(1)} \end{bmatrix} = \begin{bmatrix} 1 \\ S_t^{(1)} \\ \vdots \\ S_t^{(d)} \end{bmatrix}$$

Using the vectorized notation introduced in Section 3.9.3.2, we can compute the hidden layer activations using:

$$\boldsymbol{a}^{(2)} = \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ \vdots \\ a_{s_2}^{(2)} \end{bmatrix} = \text{sigmoid}\left(\boldsymbol{\theta}^{(1)}\boldsymbol{a}^{(1)}\right),$$

where s_2 is the total number of hidden units, and $\boldsymbol{\theta}^{(1)} \in \mathbb{R}^{s_2 \times d+1}$ representing the connecting weights between the input and hidden layers (the $d+1$st term is for bias correction). To finalize forward propagation needed to determine a control u_t given an input state $\boldsymbol{S}_t$, we append $\boldsymbol{a}_2$ by $a_0^{(2)} = 1$ and compute:

$$u_t = \boldsymbol{a}^{(3)} = \begin{bmatrix} a_1^{(3)} \\ a_2^{(3)} \\ \vdots \\ a_m^{(3)} \end{bmatrix} = \text{sigmoid}\left(\boldsymbol{\theta}^{(2)}\text{sigmoid}\left(\boldsymbol{\theta}^{(1)}\boldsymbol{a}^{(1)}\right)\right).$$

At this stage, we are ready to plug-in the above policy representation in our optimization problem. To write this in the notational style we have used elsewhere in the chapter, we are going to let $\theta = \boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}$, allowing us to write our problem as

$$\min_{\theta} F^{\pi}(\theta) = \mathbb{E}\left[\sum_{t=1}^{T} C(S_t, U^{\pi}(S_t|\theta))\right] + \frac{\lambda}{2}\sum_{l=1}^{2}\left[\sum_{i=1}^{s_l}\left[\sum_{j=1}^{s_{l+1}}\left(\boldsymbol{\theta}_{ji}^{(l)}\right)^2\right]\right], \quad (12.37)$$

where $U^{\pi}(S_t|\theta) = \text{sigmoid}\left(\boldsymbol{\theta}^{(2)}\text{sigmoid}\left(\boldsymbol{\theta}^{(1)}\boldsymbol{a}^{(1)}\right)\right)$ is the policy. The term $\sum_{l=1}^{2}\left[\sum_{i=1}^{s_l}\left[\sum_{j=1}^{s_{l+1}}\left(\boldsymbol{\theta}_{ji}^{(l)}\right)^2\right]\right]$ is a regularizer used to avoid overfitting, where λ scales the regularizer (we first saw this in section 3.7.2 when we introduced Lasso).

Equation (12.37) is not easy to solve because the objective is nonconvex. We can overcome this in low dimensions by performing a series of random restarts, and then performing gradient search steps (such as those presented in chapter 7). With neural networks, however, the number of parameters can be quite large, leaving us with little more than a hope that we have found a good solution. Solving these problems remains an active area of research.

Learning model parameters corresponds to determining $\theta = \boldsymbol{\theta}^{(1)}$ and $\boldsymbol{\theta}^{(2)}$ to minimize the optimization problem above. The problem that arises when using neural networks is that $F^{\pi}(\theta)$ is nonconvex. The most widely used strategy is to start from a randomized value of θ^k and then used gradient methods to find a local minimum. This is then repeated K times, after which we keep the θ^k that performs the best. It is necessary to experiment with different sample sizes K to find a value that strikes a good tradeoff between compute time and solution quality.

Before being able to compute the gradients, however, we require a procedure allowing us to estimate the objective function $F^{\pi}(\theta)$ since we cannot compute the expectation. We can use two approaches for optimizing the parameters. The first (and easiest) is to use classical stochastic gradient search based on our (now familiar) updating equation (12.7), which involves simulating the policy and computing gradients directly from the simulation using equations (12.35) and (12.36).

The second approach is to use batch learning. We can approximate the expectation in (12.37) by replacing it using a sample $\omega^1, \ldots, \omega^N$ where each ω^n refers to a particular sample realization of $W_1, \ldots, W_T$. This allows us to rewrite the optimization problem in (12.37) as

$$\min_{\theta=\boldsymbol{\theta}^{(1)},\boldsymbol{\theta}^{(2)}} \underbrace{\frac{1}{N}\sum_{n=1}^{N}\left[\sum_{t=1}^{T} C\left(S_t\left(\omega^n\right), U_t^{\pi}(S_t|\theta)\right)\right]}_{F^{\pi}(\theta)} \quad (12.38)$$

$$\underbrace{+\frac{\lambda}{2}\sum_{l=1}^{2}\left[\sum_{i=1}^{s_l}\left[\sum_{j=1}^{s_{l+1}}\left(\boldsymbol{\theta}_{ji}^{(l)}\right)^2\right]\right]}_{\text{Regularization term}}. \quad (12.39)$$

Readers should recognize this as nothing more than an example of a sampled approximation of an expectation, as we discussed in section 4.3. This is a deterministic optimization problem, but that does not mean it is easy. Designing a solution strategy depends on the structure of the cost function $C(S_t, u_t)$, the policy $U^{\pi}(S_t|\theta)$, and the transition function $S_{t+1} = S^M(S_t, u_t, W_{t+1})$.

12.6 DERIVATIVE-FREE POLICY SEARCH

Derivative-free policy search involves generating a population of parameters $\theta_1, \ldots, \theta_K$ and then searching for the best one using the tools that we presented in chapter 7. Since the vector θ is continuous, some methods are more appropriate than others.

Below, we highlight a tour through chapter 7 on derivative-free methods, with a brief visit to chapter 5. We do this by first discussing the role of dimensionality of the control vector θ, then list some of the more appropriate belief models, and close by touching on the issue of optimizing final or cumulative rewards.

12.6.1 The role of dimensionality

It is important to understand the role of dimensionality in the vector θ. There are three major classes:

- Scalar models - Boltzmann policies are characterized by a scalar parameter θ, which makes it relatively easy to search, especially if we can run long (and relatively low noise) simulations. We can typically discretize the range for θ and then use any of the methods described in chapter 7 for discrete alternatives with lookup table belief models. Since we are maximizing a continuous function, it makes sense to exploit correlated beliefs, first introduced in section 3.4.2, and exploited in the knowledge gradient for correlated beliefs in section 7.8.1.

- Low-dimensional continuous models - Imagine a problem with a relatively simple state variable (the amount of water in a reservoir, or indices for the stock market and interest rates). If there are only two or three dimensions, we may still be able to enumerate a full grid and apply the techniques for lookup tables (presumably exploiting correlated beliefs). If there are more than three dimensions, enumerating a discretized grid tends to be impractical, but we can begin using classical parametric models, as described in section 7.8.3 for linear belief models. Alternatively, if we have a nonlinear model, we could use sampled belief models as described in section 7.8.3.

- High-dimensional continuous models - A good example of a high-dimensional policy is a neural network, but it could also arise with a linear model with a high-dimensional state variable (as might arise in the management of complex resources). These models have typically been approached using gradient methods (described above), so there is little research in the optimization of high-dimensional policies using derivative-free methods. For example, we could use a sampled belief model as described in section 7.8.3, which might involve creating a set $\theta^1, \ldots, \theta^K$ for $K = 20$ or so, for a parameter vector θ that may have hundreds of dimensions.

12.6.2 Belief models

We can draw on a number of the different belief structures presented in chapter 3. Some that are likely to be useful in the representation of continuous vectors for the parameter vector θ include:

- Lookup table with correlated beliefs - This could work well for vectors θ with one to three dimensions.

- Low-dimensional linear models (e.g. quadratic) - Low dimensional linear models can be used in a number of settings, spanning anywhere from one to dozens of variables.

- Sparse linear models - These models extend the linear models to the domain of high-dimensional vectors, but where we think that many of the elements of θ may be zero.

- Neural networks - We have described gradient-based search models using neural network policies (which can be very high dimensional), but we may not be able to use these methods if we cannot differentiate the cost function or the transition function, possibly because one or both are not known (this is more likely to be the case with the transition function).

- Sampled belief model - Here we generate a sample $\theta^1, \ldots, \theta^K$, and then work to learn which one of these produces the best performance. Sampled belief models can be used in principle for virtually any setting, although we have no experience using this for high-dimensional problems.

Finally, we can always draw on the idea of using finite differences, which we introduced in section 5.4.3 in our chapter for derivative-based stochastic search. Although finite differences simply represent a method for approximating gradients, they are actually a derivative-free method and can be used for problems where we cannot extract a gradient as we outlined above.

12.6.3 Cumulative vs. terminal rewards

It is easy to envision settings where policies are optimized offline in a simulated environment, where we are willing to make mistakes to find the best policy, or online in a field situation, where we need to pay attention to its performance as we proceed. We handled these two settings under the names of terminal reward (for offline learning) and cumulative reward (for online learning) in chapter 7 for derivative-free stochastic search, where it is possible to search for policies using either objective function. However, as of this writing, we are unaware of an adaptation of derivative-based stochastic search for online (cumulative reward) settings.

12.7 SUPERVISED LEARNING

An entirely different approach to policy search is to take advantage of the presence (if available) of an external source of decisions. This might be a domain expert (such as a doctor making medical decisions, experienced air traffic control dispatchers, or drivers operating a car), or perhaps simply a different optimization-based policy. For example, in chapter 20, we are going to introduce lookahead policies that require forecasting into the future and solving what can become a very large optimization problem, possibly one that is too large to solve in practice.

Imagine that we have a set of decisions x^n from an external source (human or computer). Let S^n be our state variable, representing information available when the nth decision was made. For the moment, assume that we have access to a dataset $(S^n, x^n)_{n=1}^N$ from past history. Now we face a classical machine learning problem of fitting a function (policy) to this data. Start by assuming that we are going to use a simple linear model of the form

$$X^\pi(S|\theta) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(S),$$

where $(\phi_f(S))_{f \in \mathcal{F}}$ is a set of features designed by a human (there is a vast machinery of statistical learning tools we can bring to bear on this problem). We can use our batch dataset to estimate $X^\pi(S|\theta)$, although more often we can use the tools in chapter 3 to adapt to new data in an online fashion.

Several issues arise when pursuing this approach:

- Our policy is never better than our supervisor, although in many cases a policy that is as good as an experienced supervisor might be quite good.
- In a recursive setting, we need to design algorithms that allow the policy to adapt as more data becomes available. Using a neural network, for example, can result in significant overfitting, producing unexpected results as the function adapts to noisy data.
- If our supervisor is a human, we are going to be limited in the number of times we can query our domain expert, raising the problem of efficiently designing questions.

Supervised learning can be a powerful strategy for finding an initial policy, and then using policy search methods (derivative-based or derivative-free) to further improve the policy. However, we face the issue of collecting data from our supervisor. If we have an extensive database of decisions and the corresponding state variables that capture the information we would use to make a decision, then we simply have a nice statistical challenge (albeit, not necessarily an easy one). However, it is often the case that we have to work with data arriving sequentially in an online manner. We can approach our policy estimation in two ways:

Active policy search Here we are actively involved in the operation of the process to design better policies. We can do this in two ways:

Active policy adjustment This involves adjusting the parameters controlling the policy, as we described above with policy search.

Active state selection We may choose the state that then determines the decision. This might be in the form of choosing hypothetical situations (e.g. patient characteristics) and then asking the expert for his/her decision.

Passive policy search In this setting, we are following some policy, and then selectively using the results to update our policy.

Active state selection is similar to derivative-free stochastic search (chapter 7). Instead of choosing x to obtain a noisy observation of $F(x) = \mathbb{E}F(x, W)$, we are choosing a state S to get a (possibly noisy) observation of an action x from some source. Active state selection can only be done in an offline setting (we cannot choose the characteristics of a patient walking into the hospital, but we can pose the characteristics of a hypothetical patient), but we are limited in terms of how many questions we can pose to our supervisor, especially if it is human (but also if it is a time consuming optimization model).

Passive policy search is an approach where we use our policy $X^\pi(S_t)$ to make decisions x_t that are then used to update the policy. Of course, if all we did was feed our own decisions back into the same function that produced the decisions, then we would not learn anything. However, it is possible to perform a weighted statistical fit, where we put a higher weight on decisions that perform better.

12.8 BIBLIOGRAPHIC NOTES

Section 5.3 - The theoretical foundation for estimating value functions from Monte Carlo estimates has its roots in stochastic approximation theory, originated by Robbins & Monro (1951), with important early contributions made by Kiefer & Wolfowitz (1952), Blum (1954*a*) and Dvoretzky (1956). For thorough theoretical treatments of stochastic approximation theory, see Wasan (1969), Kushner & Clark (1978) and Kushner & Yin (1997). Very readable treatments of stochastic optimization can be found in Pflug (1996) and **?**.

Section **??** - A nice introduction to various learning strategies is contained in Kaelbling (1993) and Sutton & Barto (1998). Thrun (1992) contains a good discussion of exploration in the learning process. The discussion of Boltzmann exploration and epsilon-greedy exploration is based on Singh et al. (2000).

Section **??** - The knowledge gradient policy for normally distributed rewards and independent beliefs was introduced by Gupta & Miescke (1996), and subsequently analyzed in greater depth by Frazier et al. (2008). The knowledge gradient for correlated beliefs was introduced by Frazier et al. (2009). The adaptation of the knowledge gradient for online problems is due to Ryzhov & Powell (2009). The knowledge gradient for correlated beliefs is due to Negoescu et al. (2010).

Section 7.9 - There is an advanced field of research within the simulation community that has addressed the problem of using simulation (in particular, discrete event simulation) to find the best setting of a set of parameters that controls the behavior of the simulation. An early survey is given by Bechhofer et al. (1995); a more recent survey can be found in Fu et al. (2007). **?** provides a nice tutorial overview of methods based on ordinal optimization. Other important contributions in this line include Hong & Nelson (2006) and Hong & Nelson (2007). Most of this literature considers problems where the number of potential alternatives is not too large. Nelson et al. (2001) considers the case when the number of designs is large. Ankenman et al. (2009) discusses the use of a technique called kriging, which is useful when the parameter vector x is continuous. The literature on optimal computing budget allocation is based on a series of articles originating with Chen (1995), and including Chen et al. (1997); **?**); **?**, and Chen et al. (2000). Chick et al. (2001) introduces the $LL(B)$ strategy which maximizes the linear loss with measurement budget B. He et al. (2007) introduce an OCBA procedure for optimizing the expected value of a chosen design, using the Bonferroni inequality to approximate the objective function for a single stage. A common strategy in simulation is to test different parameters using the same set of random numbers to reduce the variance of the comparisons. Fu et al. (2007) apply the OCBA concept to measurements using common random numbers.

Section 5.8.2 - This proof is based on Blum (1954*b*), which generalized the original paper by Robbins & Monro (1951).

Section 5.8.3 - The proof in section 5.8.3 uses standard techniques drawn from several sources, notably Wasan (1969), Chong (1991), Kushner & Yin (1997) and, for this author, Powell & Cheung (2000).

PROBLEMS

12.1

CHAPTER 13

COST FUNCTION APPROXIMATIONS

Parametric function approximations (chapter 12) can be a particularly powerful strategy for problems where there is a clear structure to the policy. For example, buying when the price is below θ^{min} and selling when it is above θ^{max} is an obvious structure for many buy/sell problems. But PFAs do not scale to larger, more complex problems such as, say, scheduling an airline or managing an international supply chain.

One way to envision CFAs is to start with a myopic (sometimes called greedy) policy where we always do what is best. This was introduced in chapter 7 for derivative-free stochastic optimization as a pure exploitation policy. Imagine that we have a discrete set of choices $\mathcal{X} = \{x_1, \ldots, x_M\}$ where μ^n_x is our estimate of some unknown function $\mathbb{E}F(x, W)$ after n samples, the pure exploitation policy would be written

$$X^{Explt}(S^n) = \arg\max_x \mu^n_x.$$

Such a policy does what appears to be best, but ignores that after choosing x^n and observing $\hat{F}^{n+1} = F(x^n, W^{n+1})$ that we can use this information to update our belief state (captured by S^n). The problem is that we may have an estimate μ^n_x that is too low that would discourage us from trying it again. One way to fix this (which we introduced in chapter 7 as interval estimation) using the modified policy

$$X^{IE}(S_t|\theta^{IE}) = \arg\max_{x\in\mathcal{X}} \big(\mu_{tx} + \theta^{IE}\sigma_{tx}\big), \tag{13.1}$$

where $S_t = (\mu^n_{tx}, \sigma^2_{tx})_{\mathcal{X}}$ is our belief state at time t, and θ is a parameter that has to be tuned through our usual objective function

$$\max_\theta F(\theta) = \mathbb{E} \sum_{t=0}^{T} C(S_t, X^\pi(S_t|\theta)). \tag{13.2}$$

We have tweaked the pure exploitation policy by adding an uncertainty bonus in (13.1) which encourages trying alternatives where μ_x might be lower, but where there is sufficient uncertainty that it might actually be higher. This is a purely heuristic way of enforcing a tradeoff between exploration and exploitation.

While our interval estimation policy is limited to discrete action spaces, parametric CFAs can actually be extended to very large-scale problems. Unlike PFAs that are generally limited to much simpler problems, a parametric CFA can be written in its most general form as

$$X^{CFA}(S_t|\theta) = \arg\max_{x \in \mathcal{X}^\pi_t(\theta)} \bar{C}^\pi(S_t, x_t|\theta), \tag{13.3}$$

where $\bar{C}^\pi(S_t, x_t|\theta)$ is a parametrically modified cost function, subject to a (possibly) modified set of constraints $\mathcal{X}^\pi(\theta)$, where θ is the vector of tunable parameters. Now imagine that $\bar{C}^\pi(S_t, x_t|\theta)$ is a concave (perhaps linear) function, where where $\mathcal{X}_t$ might be a set of linear constraints such as

$$\mathcal{X}_t = \{x | A_t x_t = b_t,\ 0 \leq x_t \leq u_t\}.$$

Now we can solve problems where x_t has many thousands (even hundreds of thousands) of variables (dimensions). Below, we even show ways of creating parametrically modified constraints to produce more robust solutions.

13.1 OPTIMAL MYOPIC POLICIES

A special case of a CFA is a myopic policy, given by

$$X^{Myopic}(S_t) = \arg\max_{x_t \in \mathcal{X}_t} C(S_t, x_t). \tag{13.4}$$

There are special cases where myopic policies are optimal. Consider, for example, a problem where the state S_t is independent of history. For example, imagine a fishing boat that returns with a catch of $\hat{R}_t$ fish caught during time interval $(t-1, t)$, which are then sold at market prices $\hat{p}_t = (\hat{p}_{tm})_{m=1}^M$ where $m = (1, \ldots, M)$ is the set of markets, each offering to purchase $(\hat{D}_{tm})_{m=1}^M$ pounds of fish. The exogenous information is $W_t = (\hat{R}_t, \hat{D}_t, \hat{p}_t)$, which then determines the state $S_t = (R_t, D_t, p_t)$ where $R_t = \hat{R}_t$, $D_t = \hat{D}_t$ and $p_t = \hat{p}_t$. There is no learning.

A myopic policy would be to solve

$$X^{Myopic}(S_t) = \arg\max_{x_t} \sum_{m=1}^{M} \hat{p}_{tm} x_{tm}, \tag{13.5}$$

subject to, for $m = 1, \ldots, M$:

$$x_{tm} \leq \hat{D}_{tm}, \tag{13.6}$$

$$x_{tm} \leq \hat{R}_{tm}, \tag{13.7}$$

$$x_{tm} \geq 0. \tag{13.8}$$

The policy $X^{Myopic}(S_t)$ defined by (13.5) - (13.8) is optimal because the decision x_t has no impact on the future. The post-decision state S_t^x is empty, since we assume that all fish are sold (or discarded), and the prices p_{t+1} and demands $\hat{D}_{t+1}$ do not depend on x_t, p_t or D_t.

Myopic policies are widely used. People purchase the cheapest product on the internet, and sell their used car to the highest bidder. A popular problem involves allocating assets in the stock market when we assume there are no transaction costs for moving money from one stock to the next. This is the same as sweeping all funds into a money market account at the end of the day and then starting over. While this problem has a physical state (the amount of money that is swept into the money market account each night) that is affected by the allocation decision made at the beginning of the day, we can still optimize our returns over time by maximizing expected returns each day.

It is easy to introduce slight modifications to any of these problems to produce a problem where the myopic policy is no longer optimal, but where it is still a good starting point. Below we consider how to improve myopic policies using parametric modifications. We then open up to a hybrid policy where we combine a deterministic lookahead with parametric modifications to improve robustness.

13.2 COST-MODIFIED CFAS

We begin by considering problems where we modify the problem through the objective function to achieve desired behaviors. Including bonuses and penalties is a widely used heuristic approach to getting cost-based optimization models to produce desired behaviors, such as balancing real costs against penalties for poor service. Not surprisingly, we can use this approach to also produce robust behaviors in the presence of uncertainty.

We begin by presenting a general way of including linear cost correction models in the objective function, and then provide an example of how this strategy can be used for a dynamic assignment problem.

13.2.1 Linear cost function correction

A generic way of improving the performance of a myopic policy is to modify (13.4) with a parametric cost function correction term consisting of a linear model, which we can write as

$$X^{CFA-cost}(S_t|\theta) = \arg\max_{x_t \in \mathcal{X}_t} \left(C(S_t, x_t) + \sum_{f \in \mathcal{F}} \theta_f \phi_f(S_t, x_t) \right). \tag{13.9}$$

where $(\phi_f)(S, x)_{f \in \mathcal{F}}$ is a set of features that depend first and foremost on x, and possibly on the state S. If a feature does not depend on the decision, then it would not affect the choice of optimal solution.

Designing the features for equation (13.9) is no different than designing the features for a linear policy function approximation (or, for that matter, any linear statistical model which we introduced in chapter 3). It is always possible to simply construct a polynomial comprised of different combinations of elements of x_t and S_t with different transformations (linear, square, ...), but many problems have very specific structure.

13.2.2 CFAs for dynamic assignment problems

The truckload trucking industry requires matching drivers to loads, just as ride-sharing companies match drivers to riders. The difference with truckload trucking is that the customer is a load of freight, and sometimes the load has to wait a while (possibly several hours) before being picked up.

A basic model of the dynamic assignment problem can be formulated using the following notation:

$\mathcal{D}_t$ = The set of all drivers (with tractors) available at time t,

$\mathcal{L}_t$ = The set of all loads waiting to be moved at time t,

S_t = $(\mathcal{D}_t, \mathcal{L}_t)$ = the state of our system at time t,

$c_{td\ell}$ = The contribution of assigning driver $d \in \mathcal{D}_t$ to load $\ell \in \mathcal{L}_t$ at time t, considering the cost of moving empty to the load, as well as penalties for late pickup or delivery,

$x_{td\ell}$ = 1 if we assign driver d to load ℓ at time t, 0 otherwise,

$\mathcal{L}^x_t$ = Set of loads that were served at time t, which is to say all ℓ where $\sum_{d\in\mathcal{D}_t} x_{td\ell} = 1$,

$\mathcal{D}^x_t$ = Set of drivers that were dispatched at time t, which is to say all d where $\sum_{\ell\in\mathcal{L}_t} x_{td\ell} = 1$.

A myopic policy for assigning drivers to loads would be formulated as

$$X^{Assign}(S_t) = \arg\max_{x_t} \sum_{d\in\mathcal{D}_t} \sum_{\ell\in\mathcal{L}_t} c_{td\ell} x_{td\ell}. \tag{13.10}$$

Once we dispatch a driver (that is, $x_{td\ell} = 1$ for some $\ell \in \mathcal{L}_t$), we assume the driver vanishes (this is purely for modeling simplification). We then model drivers becoming available as an exogenous stochastic process along with the loads. This is modeled using

$\hat{L}_t$ = Exogenous process describing random loads (complete with origins and destinations) that were called in between $t-1$ and t,

$\mathcal{D}_t$ = Exogenous process describing drivers calling in to say they are available (along with location).

In practice $\mathcal{D}_t$ will depend on prior decisions, but this simplified model will help us make the point. The transition function would be given by

$$\mathcal{L}_{t+1} = \mathcal{L}_t \setminus \mathcal{L}^x_t \bigcup \hat{L}_{t+1}, \tag{13.11}$$

$$\mathcal{D}_{t+1} = \mathcal{D}_t \setminus \mathcal{D}^x_t \bigcup \hat{D}_{t+1}. \tag{13.12}$$

We can modify our updating logic to also drop loads that have been waiting too long.

The problem with our myopic policy in (13.10) is that a load may not be moved at time t, which means it is still waiting to be moved at time $t+1$. There are different ways of handling this, but an obvious one is to put a positive bonus for moving loads that have been delayed. Let

$\tau_{t\ell}$ = The time that load $\ell \in \mathcal{L}_t$ has been delayed as of time t.

Now consider the modified policy

$$X^{CFA-Assign}(S_t|\theta) = \arg\max_{x_t} \sum_{d\in\mathcal{D}_t}\sum_{\ell\in\mathcal{L}_t}(c_{td\ell} + \theta\tau_{t\ell})x_{td\ell}. \tag{13.13}$$

Now we have a modified cost function (we use the term "cost function" even though we are maximizing) that is parameterized by θ. The next challenge is to tune θ: too large, and we move long distances to pull loads that have been waiting, too small, and we end up losing loads that have to wait too long. Our optimization problem is given by

$$\max_\theta \mathbb{E}\sum_{t=0}^{T} C(S_t, X^{CFA-Assign}(S_t|\theta)) \tag{13.14}$$

where

$$C(S_t, x_t) = \sum_{d\in\mathcal{D}_t}\sum_{\ell\in\mathcal{L}_t} c_{td\ell}x_{td\ell}.$$

This is a classical use of a parametric cost function approximation for finding robust policies for a very high-dimensional resource allocation problem. The delay penalty parameter θ can be tuned in a simulator that represents the objective (13.14) along with the dynamics (13.11) and (13.12). In real applications, this tuning is often done (albeit in an ad hoc way) in an online setting based on real observations.

13.2.3 Policy search for cost-modified CFAs

Considerable caution has to be applied if you want to use a stochastic gradient method for optimizing cost-modified CFAs since the objective function $F(\theta)$ (see equation 13.2) is generally not going to be differentiable with respect to θ. Small changes in θ may produce sudden jumps, with intervals where there is no change at all. A more promising approach is probably to generate a sample $\theta_1, \ldots, \theta_K$ of parameter values, and then to use the methods of derivative-free stochastic optimization from chapter 7.

As of this writing we have limited experience with cost-modified CFAs, but the few cases in which we have seen it used, it has worked quite well, especially when compared to policies based on value function approximations with the exact same structure. We will return to these results in chapter 18 on approximate dynamic programming.

13.3 CONSTRAINT-MODIFIED CFAS

A particularly powerful solution strategy, widely used in industry, is to modify the constraints of a problem to produce more robust solutions. Some examples where this approach has been used are listed below.

■ **EXAMPLE 13.1**

Airlines routinely use deterministic scheduling models to plan the movements of aircraft. Such models have to be designed to represent the travel times between cities, which can be highly uncertain. To handle this, the model inserts schedule slack θ_{ij} for city pair (i, j). These parameters can then be optimized in a simulator, but in practice they are optimized in the field, striking a balance between equipment productivity (which encourages smaller values of θ) and penalties for late arrivals (which encourages larger values of θ).

■ **EXAMPLE 13.2**

A major retailer has to manage inventories for a long supply chain extending from the far East to North America. Uncertainties in production and shipping require that the retailer maintain buffer stocks. Let θ be the amount of buffer stock planned in the future (inventory is allowed to go to zero at the last minute). The buffer stock θ enters through the constraint, and has to be optimized in a simulator or online in the field.

■ **EXAMPLE 13.3**

Independent system operators (ISOs) have to plan how much energy to generate tomorrow based on a forecast of loads, as well as energy to be generated from wind and solar. They used a forecast factored by a vector θ with elements for each type of forecast.

Constraint-modified CFAs can be written in the form

$$X^{Con-CFA}(S_t|\theta) = \arg\max_{x_t} C(S_t, x_t), \tag{13.15}$$

subject to

$$\begin{aligned} A_t \tilde{x}_t &= b(\theta), && (13.16) \\ \tilde{x}_t &\geq 0. && (13.17) \end{aligned}$$

where

$$b(\theta) = b \otimes \theta^b + D\theta^c$$

is the element by element product of the vector b with the similarly dimensioned vector of coefficients θ^b, while D is a suitably constructed matrix that translates θ^c (which may be scalar or a vector). We then parameterize our policy by $\theta = (\theta^c, \theta^b)$.

To illustrate, consider a simplified version of a problem for planning energy systems. In our streamlined version, an independent system operator (ISO) which manages energy resources over the grid has to plan which generators to turn on 24 hours in advance based on a forecast. Steam generators have to be planned at least 10 hours in advance because of the amount of time it takes to boil large tanks of water; for this reason, these problems are typically solved the day before, using forecasts for loads (which are affected by weather), as well as energy from wind and solar.

D_t = Demand ("load") for power during hour t,
$f^D_{tt'}$ = Forecast of $D_{t'}$ made at time t,
E_t = Energy generated from renewables (wind/solar) during hour t,
R_t = Amount of energy stored in the battery at time t,
$f^E_{tt'}$ = Forecast of $E_{t'}$ made at time t,
$u_{tt'}$ = Limit on how much generation can be committed at time t for hour t',
$c_{tt'}$ = Price to be paid for energy generated during hour t' when the commitment is made at time t.

The decision variables are given by

$\tilde{x}_{tt'}$ = Planned generation of energy during hour t', where the plan is made at time t, which is comprised of the following elements:
$\tilde{x}^{ED}_{tt'}$ = Flow of energy from renewables to demand,
$\tilde{x}^{EB}_{tt'}$ = Flow of energy from renewables to battery,
$\tilde{x}^{GD}_{tt'}$ = Flow of energy from grid to demand,
$\tilde{x}^{GB}_{tt'}$ = Flow of energy from grid to battery,
$\tilde{x}^{BD}_{tt'}$ = Flow of energy from battery to demand.

We have to create projections of the energy in the battery over the horizon $t' > t$:

$$R_{t+1,t'} = R_{tt'} + \tilde{x}^{EB}_{tt'} + \tilde{x}^{GB}_{tt'} - \tilde{x}^{BD}_{tt'}.$$

The estimate $R_{t+1,t+1}$ becomes the actual energy in the battery as of time $t+1$, while $R_{t+1,t'}$ for $t' \geq t+2$ are projections that may change.

The state variable is now given by

$$S_t = (R_t, (f^D_{tt'})_{t' \geq t}, (f^E_{tt'})_{t' \geq t}).$$

We are going to ignore any linking between hours (for example, if we are generating a small quantity of energy during hour t, it is hard to suddenly ramp this up quickly and generate a lot during hour $t+1$). Our constraint-CFA policy might look like

$$X^{Con-CFA}_t(S_t|\theta) = \mathop{\arg\min}_{x_t, \tilde{x}_{t,t+1}, \ldots, \tilde{x}_{t,t+H}} \sum_{\bar{t}=t}^{t+H} \tilde{c}_{tt'} \tilde{x}_{tt'}, \tag{13.18}$$

subject to (for all $t' > t$):

$$\begin{aligned}
\sum_{t'=t}^{t+H} \left(\tilde{x}^{ED}_{tt'} + \tilde{x}^{GD}_{tt'} + \tilde{x}^{BD}_{tt'}\right) &= \theta^D_{t'-t} f^D_{tt'}, && (13.19)\\
\sum_{t'=t}^{t+H} \left(\tilde{x}^{ED}_{tt'} + \tilde{x}^{EB}_{tt'}\right) &\leq \theta^E_{t'-t} f^E_{tt'}, && (13.20)\\
\tilde{x}^{BD}_{tt'} &\leq \tilde{R}_{tt'}, && (13.21)\\
\tilde{x}^{EB}_{tt'} + \tilde{x}^{GB}_{tt'} + \tilde{R}_{tt'} &\leq R^{max}, && (13.22)\\
& && (13.23)
\end{aligned}$$

We can evaluate the performance of the policy while following sample path $W_1(\omega), \ldots, W_t(\omega), \ldots, T)$ using

$$F(\theta, \omega) = \sum_{t=0}^{T} c_t X_t^{Con-CFA}(S_t|\theta), \tag{13.24}$$

where $S_{t+1}(\omega) = S^M(S_t(\omega), X_t^{Con-CFA}(S_t(\omega)|\theta), W_{t+1}(\omega))$ is the sequence of states.

We can write the constraints in matrix-vector form using

$$\begin{aligned} A_t \tilde{x}_t &\leq b_t(\theta), \\ \tilde{x}_t &\geq 0. \end{aligned}$$

13.4 CFA FOR RESOURCE ALLOCATION PROBLEMS

In this section, we consider decisions x_t which represent quantities, as in flows of water, energy, money or drivers. This is going to change how we capture the impact of a decision now on the future.

13.4.1 The model

Resource allocation problems feature a resource state vector $R_t = (R_{ti})_{i \in \mathcal{I}}$ where R_{ti} might be the amount of water in reservoir i, the amount of blood of type i, or the number of cars where i might capture both the model and the location of the inventory. We act on these resources with a vector x_t, which can capture decisions which increase or decrease the inventory. Then, we may observe random perturbations $\hat{R}_{t+1,i}$ which could represent rainfall (for a reservoir), blood donations, or unexpected changes in inventory (late deliveries, sales, theft). We assume that our resource vector R_t evolves according to

$$R_{t+1} = R_t + A_t x_t + \hat{R}_{t+1},$$

where A_t is a suitable dimensioned matrix. For example, the matrix A_t might consist of 0's, 1's and -1's to indicate whether an element of the vector x_t affects (increases or decreases) the inventory of a particular type. We can easily introduce logic to ensure that no element of R_{t+1} goes negative.

We are going to assume that our state variable $S_t = R_t$, although in practice the state variable might include information other than R_t. As above, we assume we have a parameterized policy $X_t^\pi(S_t|\theta)$ that might involve solving a linear program parameterized by θ. For example, imagine that we are managing blood where we have to satisfy the demand D_{ti} for blood type $i \in \mathcal{I}$ at time t. A simple myopic policy would be

$$\max_{x_{tij},\, i,j \in \mathcal{I}} \sum_{i \in \mathcal{I}} \left(\sum_{j \in \mathcal{I}} c_{ij} x_{tij} - M s_{ti} \right), \tag{13.25}$$

subject to

$$\begin{aligned} \sum_{j \in \mathcal{I}} x_{tij} &\leq R_{ti},\ \forall i, && (13.26) \\ \sum_{i \in \mathcal{I}} x_{tij} + s_{ti} &= D_{tj},\ \forall j, && (13.27) \\ x_{tij}, s_{ti} &\geq 0,\ \forall\, i, j. && (13.28) \end{aligned}$$

We assume the costs capture our ability to substitute blood (using blood type i for a demand D_{tj}), with high costs when this is not allowed. We also have a penalty for not satisfying demand. We do not have to assign all of our blood in R_{ti}; anything not assigned will be held over.

This is a pretty basic model. We might want to assign blood to handle not just the demand D_{tj}, but also forecasted demands. Let f^D_{tj} be our forecast of the demand of blood type j over the next four weeks. But, we might not trust our forecast, so we modify the forecast to θf^D_{tj}. We would then replace (13.27) with

$$\sum_{i\in\mathcal{I}} x_{tij} + s_{ti} \quad = \quad D_{tj} + \theta f^D_{tj},\ \forall j, \tag{13.29}$$

$$\tag{13.30}$$

We can refer to the constraints (13.26), (13.28) and (13.29) as the set $\mathcal{X}_t(\theta)$. We can then write (13.25) in the form of a policy as

$$X^\pi_t(S_t|\theta) = \arg\max_{x_t\in\mathcal{X}_t(\theta)} \sum_{i\in\mathcal{I}} \left(\sum_{j\in\mathcal{I}} c_{ij}x_{tij} - Ms_{ti}\right), \tag{13.31}$$

This is a sample of a parameterized cost function approximation, which we revisit in depth in

13.4.2 Policy gradients

Adapting from (12.35) and (12.36), we can derive the gradient with respect to the CFA policy to obtain

$$\begin{aligned}\nabla_\theta F^\pi(\theta,\omega) \;=\; & \left(\frac{\partial C_0(S_0,x_0)}{\partial x_0}\right) + \sum_{t'=1}^{T}\left[\left(\frac{\partial C_{t'}(S_{t'},X^\pi_{t'}(S_{t'}))}{\partial S_{t'}}\frac{\partial S_{t'}}{\partial \theta}\right)\right.\\ & \left.+\frac{\partial C_{t'}(S_{t'},x_{t'})}{\partial x_{t'}}\left(\frac{\partial X^\pi_{t'}(S_{t'}|\theta)}{\partial S_{t'}}\frac{\partial S_{t'}}{\partial \theta} + \frac{\partial X^\pi_{t'}(S_{t'}|\theta)}{\partial \theta}\right)\right]\end{aligned} \tag{13.32}$$

where

$$\frac{\partial S_{t'}}{\partial \theta} = \frac{\partial S_{t'}}{\partial S_{t'-1}}\frac{\partial S_{t'-1}}{\partial \theta} + \frac{\partial S_{t'}}{\partial x_{t'-1}}\left[\frac{\partial X^\pi_{t'-1}(S_{t'-1}|\theta)}{\partial S_{t'-1}}\frac{\partial S_{t'-1}}{\partial \theta} + \frac{\partial X^\pi_{t'-1}(S_{t'-1})}{\partial \theta}\right] \tag{13.33}$$

13.5 BIBLIOGRAPHIC NOTES

- Section xx -

PART V - LOOKAHEAD POLICIES

Lookahead policies are based on estimates of the impact of a decision on the future. One way to do this is by creating a value function, which captures the value of being in a state. This can be used to help us understand the downstream impact of a decision made now. If we can find accurate estimates of these values (which can be a lot of work), the resulting policies can be quite easy to use. But there are many situations where these functions cannot be estimated accurately, in which case we have to resort to direct lookahead policies (DLAs), which are the most brute force of all the policies.

Policies based on value functions have attracted considerable attention over the years. In fact, terms like "dynamic programming" and "optimal control" are basically synonymous with value functions (or cost-to-go functions, as they are known in control theory).

Value functions are part of the broader strategy of creating policies based on approximating the downstream impact of decisions made now. This starts with the original optimization problem, which we can write in terms of finding the value when we start with initial state S_0:

$$V_0(S_0) = \max_{\pi\in\Pi} \mathbb{E}^\pi \left\{ \sum_{t'=0}^{T} C(S_{t'}, X^\pi_{t'}(S_{t'}))|S_0 \right\}. \tag{13.34}$$

We can then rewrite the problem start at time t when we are in state S_t:

$$V_t(S_t) \quad = \quad \max_{\pi\in\Pi} \mathbb{E}^\pi \left\{ \sum_{t'=t}^{T} C(S_{t'}, X^\pi_{t'}(S_{t'}))|S_t \right\}. \tag{13.35}$$

If we could compute the value function, we would be able to form a policy using

$$X^\pi *_t (S_t) \quad = \quad \arg\max_{x_t\in\mathcal{X}_t} \left(C(S_t, x_t) + \mathbb{E}\left\{V_{t+1}(S_{t+1})|S_t\right\}\right). \tag{13.36}$$

If the state variable S_t is discrete with not too many values, we can use the tools of discrete Markov decision processes, which are introduced in chapter 14. These techniques suffer from what is widely known as the "curse of dimensionality," although in reality there are three curses (states, actions and the exogenous information).

When problems become more difficult, we have to turn to approximation methods. We begin in chapter 16 by introducing the idea of *backward approximate dynamic programming*, which works the same way as the techniques we introduce in chapter 14, but they scale to problems with somewhat larger state spaces (but still restricted to discrete action spaces).

A different class of methods is known as *forward approximate dynamic programming*, which can scale to truly large scale problems such as optimizing a fleet of thousands of trucks or locomotives. These methods are developed for general problems over chapters 17, where we show how to estimate value functions for a fixed policy, and 18, where we address the much harder problem of finding good (ideally near optimal) policies. Chapter 19 describes forward approximate dynamic programming for the important special case of convex problems, which arise in a wide range of resource allocation problems.

Whether we use backward or forward approximate dynamic programming, we end up with value function approximations that we call $\overline{V}_t(S_t)$, from which we can derive a policy that looks like

$$X_t^{VFA}(S_t) \quad = \quad \arg\max_{x_t \in \mathcal{X}_t} \left(C(S_t, x_t) + \mathbb{E}\left\{ \overline{V}_{t+1}(S_{t+1}) | S_t \right\} \right). \tag{13.37}$$

One problem that often arises in using (13.37) is that we may not be able to compute the expectation. There are many applications where we can avoid this by using the concept of the post-decision state, which we described in some depth in section 9.3.4. This allows us to write the policy in the form

$$X_t^{VFA}(S_t) \quad = \quad \arg\max_{x_t \in \mathcal{X}_t} \Big(C(S_t, x_t) + \overline{V}_t^x(S_t^x) \Big). \tag{13.38}$$

In fact, it is this problem structure that has allowed us to solve some ultra-large scale problems, such as optimizing a fleet of 7,000 trucks or the entire fleet of locomotives for a major railroad.

Value function approximations are a powerful algorithmic strategy, but as with everything else, they are not a panacea. They work when a problem lends itself to approximating the value in a state.

There are a number of settings where value function approximations do not work well, but one important class of problems are those that involve planning now with a forecast of the future. Forecasts are notoriously difficult to handle using value functions. These are problems that are best suited to the most brute force of all policies, direct lookaheads, or DLAs, which we discuss in depth in chapter 20.

A simple example of a direct lookahead policy is when your navigation system plans your path based on estimates of travel times over a network. This would be a deterministic lookahead, which is widely used in engineering practice. Or, we can turn to a full-blown stochastic lookahead, which produces the frightening equation

$$X_t^{\pi^*}(S_t) \quad = \quad \arg\min_{x_t \in \mathcal{X}_t} \left(C(S_t, x_t) + \mathbb{E}\left\{ \min_{\pi \in \Pi} \mathbb{E}^{\pi} \left\{ \sum_{t'=t+1}^{T} C(S_{t'}, X_{t'}^{\pi}(S_{t'})) S_{t+1} \right\} | S_t, x_t \right\} \right). \tag{13.39}$$

Just as exact value functions can be impossible to compute, equations such as (13.39) can also be impossible to compute, but chapter 20 will describe several classes of approximations.

CHAPTER 14

DISCRETE MARKOV DECISION PROCESSES

A particularly important class of problems that capture decisions under uncertainty are known as discrete Markov decision processes, which are characterized by a (not too large) set of discrete states, and a (not too large) set of discrete actions. While the “not too large” requirement limits the range of applications, this still captures a rich set of applications. Perhaps even more important is that the study of this problem class has helped to establish the theory of sequential decision problems, and has laid the foundation for different algorithmic strategies even when the assumption of small state and action spaces does not apply.

To understand the power of the Markov decision process framework, it is useful to return to the idea of a decision tree, illustrated in figure 14.1. We enumerate the decisions out of each decision node (squares), and the random outcomes out of each outcome node (circles). If there are 10 possible decisions and 10 possible random outcomes, our tree is 100 times bigger after one sequence of decisions and random information. If we step forward 10 steps (10 decisions followed by random information), our tree would have 100^{10} ending nodes. And this is not even a large problem (it is easy to find problems with far larger numbers of actions and outcomes). The explosive growth in the size of the decision trees is illustrated in figure 14.1, where the number of decisions and outcomes is quite small.

The breakthrough of Markov decision processes was the recognition that each decision node corresponds to a state of a dynamic system. In the classical representation of a decision tree, decision nodes correspond to the entire history of the process up to that point in time. However, there are many settings where we may not need to know the entire history. Assume instead that the relevant information we need to make a decision can be represented by a state s that falls in a discrete set $\mathcal{S} = (1, 2, \ldots, |\mathcal{S}|)$, where $\mathcal{S}$ is

Stochastic Optimization and Learning. By Warren B. Powell

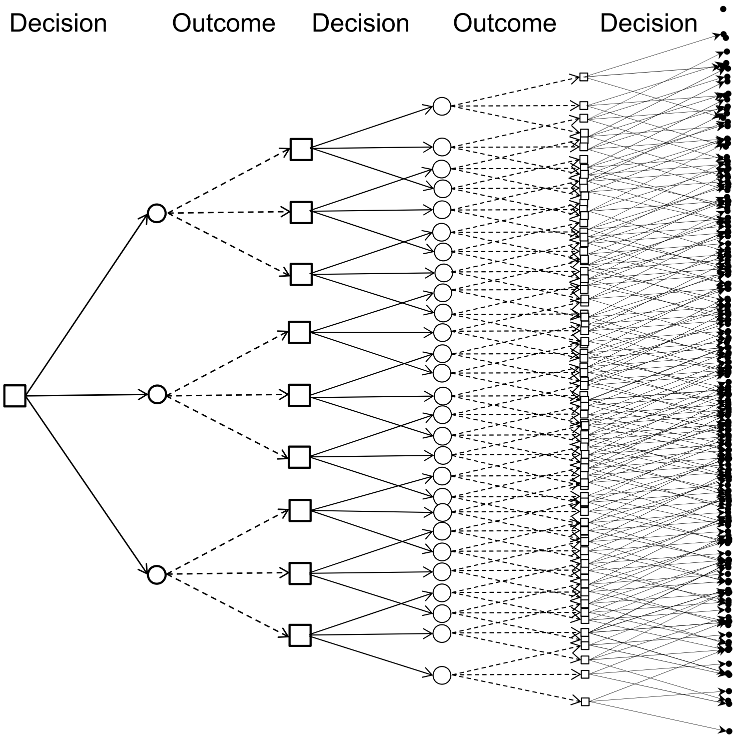


Figure 14.1 Decision tree illustrating the sequence of decisions and new information, illustrating the explosive growth of decision trees.

small enough to enumerate. For example, S_t might be the number of units of blood in a hospital inventory. In this case, the number of decision nodes does not grow exponentially. Furthermore, we only need to know the inventory, and not the history of how we got there.

When we can exploit this more compact structure, our decision tree collapses into the diagram shown in figure 14.2, where the number of states in each period is fixed. Note that the number of outcome nodes is potentially quite large. In fact, there are problems where the number of outcomes is itself quite large (for example, our random information may be continuous or multidimensional (this would be the second of the three curses of dimensionality we first introduced in section 2.1.9).

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, 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. In fact, it is often the case that the tools of Markov decision processes offers the only path to finding the *optimal* policy. Second, we can use optimal policies, which are limited to fairly small problems, to evaluate approximation algorithms that can be scaled to larger problems. Third, the theory of Markov decision processes can be used to identify structural properties that can help us identify properties of optimal policies that we can exploit in policy search algorithms. And fourth, this material provides the intellectual foundation

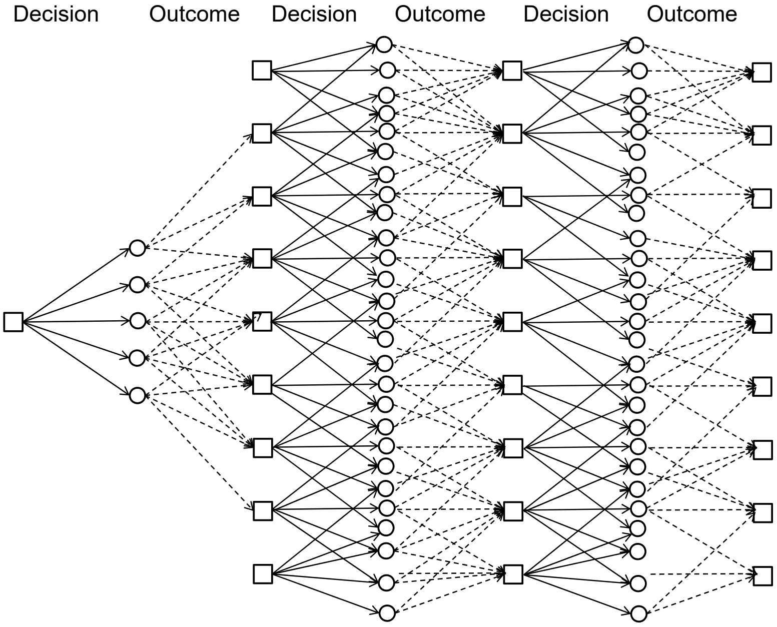


Figure 14.2 Collapsed version of the decision tree, when states do not capture entire history.

for approximation algorithms that can be scaled to far more complex problems, such as optimizing the locomotives for a major railroad, or optimizing a network of hydroelectric reservoirs.

As with most of the chapters in the book, the body of this chapter focuses on the algorithms. Some of the elegant theory that has been developed for this field is presented in the "Why does it work" section (section 14.10). 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.

14.1 THE OPTIMALITY EQUATIONS

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

$$\max_{\pi} \mathbb{E} \left\{ \sum_{t=0}^{T} \gamma^t C(S_t, A_t^{\pi}(S_t)) \right\}. \tag{14.1}$$

The most important contribution of the material in this chapter is that it provides a path to optimal policies. In practice, optimal policies are rare, so even with the computational limitations, having at least a framework for characterizing optimal policies is exceptionally valuable.

14.1.1 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 γ. 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

$$\begin{aligned} V_t(S_t) &= \max_{a_t \in \mathcal{A}_t} \big(C_t(S_t, a_t) + \gamma V_{t+1}(S_{t+1}(S_t, a_t))\big) \\ &= C_t(S_t, a_t^*(S_t)) + \gamma V_{t+1}(S_{t+1}(S_t, a_t^*(S_t))). \end{aligned} \tag{14.2}$$

Equation (14.2) 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 (14.2) by simply adding an expectation, giving us

$$V_t(S_t) \quad = \quad \max_{a_t \in \mathcal{A}_t} \big(C_t(S_t, a_t) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(S_{t+1} = s' | S_t, a_t) V_{t+1}(s') \big). \qquad (14.3)$$

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

$$V_t(S_t) \quad = \quad \max_{a_t \in \mathcal{A}_t} \big(C_t(S_t, a_t) + \gamma \mathbb{E}\{V_{t+1}(S_{t+1}(S_t, a_t, W_{t+1})) | S_t\} \big), \qquad (14.4)$$

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 (14.4) is often written in the slightly more compact form

$$V_t(S_t) \quad = \quad \max_{a_t \in \mathcal{A}_t} \big(C_t(S_t, a_t) + \gamma \mathbb{E}\{V_{t+1}(S_{t+1}) | S_t\} \big), \qquad (14.5)$$

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 (14.4) 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.

$$V_t(S_t) \quad = \quad \max_{a_t \in \mathcal{A}_t} \big(C_t(S_t, a_t) + \gamma \mathbb{E}\{V_{t+1}(S_{t+1}) | S_t, a_t\} \big) \qquad (14.6)$$

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

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

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 (14.6) 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 (14.3) 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 π 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^π for each policy (decision rule) π.

Now let c_t^π 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 (14.3) is equivalent to

$$\begin{bmatrix} \vdots \\ v_t(s) \\ \vdots \end{bmatrix} = \max_\pi \left(\begin{bmatrix} \vdots \\ c_t^\pi(s) \\ \vdots \end{bmatrix} + \gamma \begin{bmatrix} \ddots & & \\ & p_{ss'}^\pi & \\ & & \ddots \end{bmatrix} \begin{bmatrix} \vdots \\ v_{t+1}(s') \\ \vdots \end{bmatrix} \right). \tag{14.7}$$

where the maximization is performed for each element (state) in the vector. In matrix/vector form, equation (14.7) can be written

$$v_t = \max_\pi \left(c_t^\pi + \gamma P_t^\pi v_{t+1} \right). \tag{14.8}$$

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 (14.8) 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 (14.1). To bring this out, we begin by writing the expected profits using policy π from time t onward

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

$F_t^\pi(S_t)$ is the expected total contribution if we are in state S_t in time t, and follow policy π 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^π recursively using

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

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 14.10.1, uses a proof by induction: assume it is true for V_{t+1}^π, and then show that it is true for V_t^π (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 (14.4) (or (14.3)). Then

$$\begin{aligned} F_t^* &= \max_{\pi\in\Pi} F_t^\pi(S_t) \\ &= V_t(S_t). \end{aligned} \tag{14.9}$$

Equation (14.9) 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 14.10.1). 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.

14.1.2 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^π is given as data (remember that there is a different matrix for each policy π). 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 Ω_{t+1} be the set of possible outcomes of W_{t+1} (for simplicity, we assume that Ω_{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 Ω_{t+1} can be much too large to enumerate. We can estimate the transition matrix statistically, but in later chapters (starting in chapter 16) 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.

14.1.3 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

$$V_t(S_t) \quad = \quad \max_{a_t} \mathbb{E}\left\{\hat{C}_{t+1}(S_t, a_t, W_{t+1}) + \gamma V_{t+1}(S_{t+1})|S_t\right\}. \tag{14.10}$$

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.

14.1.4 Bellman's equation using operator notation*

The vector form of Bellman's equation in (14.8) can be written even more compactly using operator notation. Let $\mathcal{M}$ be the "max" (or "min") operator in (14.8) that can be viewed as acting on the vector v_{t+1} to produce the vector v_t. If we have a given policy π, 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

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

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

$$\mathcal{M}^\pi(v) = c_t^\pi + \gamma P^\pi v \tag{14.11}$$

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 14.10), 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.

14.2 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 14.3, 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 $S_t \in \mathcal{S}$.

We first saw backward dynamic programming in section 20.3.3 when we described a simple decision tree problem. The only difference between the backward dynamic programming algorithm in figure 14.3 and our solution of the decision tree problem is

```
Step 0. Initialization:
        Initialize the terminal contribution $V_T(S_T)$.
        Set $t = T - 1$.
Step 1a. Step backward in time $t = T, T-1, \dots, 0$:
    Step 2a. Loop over states $s \in \mathcal{S} = \{1, \dots, |\mathcal{S}|\}$:
    Step 2b. Initialize $V_t(s) = -M$ (where $M$ is very large).
        Step 3a. Loop over each action $a \in \mathcal{A}(s)$:
            Step 4a Initialize $Q(s,a) = 0$.
            Step 4b. Find the expected value of being in state $s$ and taking action $a$:
            Step 4c. Compute $Q_t(s,a) = \sum_{w \in \mathcal{W}} \mathbb{P}(w|s,a) V_{t+1}(s' = s^M(s,a,w))$.
            Step 4c. If $Q_t(s,a) > V_t(s)$ then
                Step 3b. Store the best value $V_t(s) = Q_t(s,a)$.
                Step 3c. Store the best action $A_t(s) = a$.
Step 1b. Return the value $V_t(s)$ and policy $A_t(s)$ for all $s \in \mathcal{S}$ and $t = 0, \dots, T$.
```

Figure 14.3 A backward dynamic programming algorithm.

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 Ω 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 ω 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 Ω 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 $V_i = \mathbb{E}\{\min_j(\hat{\tau}_{ij} + V_j)\}$.

14.3 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

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

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 optimality equations

$$V(s) \quad = \quad \max_{a \in \mathcal{A}} \left\{ C(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s,a)V(s') \right\}. \tag{14.12}$$

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

$$\max_{\pi \in \Pi} \mathbb{E} \left\{ \sum_{t=0}^{\infty} \gamma^t C_t(S_t, A_t^\pi(S_t)) \right\}. \tag{14.13}$$

Now define

$$\begin{aligned} P^{\pi,t} \quad &= \quad t\text{-step transition matrix, over periods } 0,1,\ldots,t-1, \text{ given policy } \pi \\ &= \quad \Pi_{t'=0}^{t-1} P_{t'}^\pi. \end{aligned} \tag{14.14}$$

We further define $P^{\pi,0}$ to be the identity matrix. As before, let c_t^π be the column vector of the expected cost of being in each state given that we choose the action a_t described by policy π, 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 π starting at time t is given by

$$v_t^\pi \quad = \quad \sum_{t'=t}^{\infty} \gamma^{t'-t} P^{\pi,t'-t} c_{t'}^\pi. \tag{14.15}$$

Assume that after following policy π_0 we follow policy $\pi_1 = \pi_2 = \ldots = \pi$. In this case, equation (14.15) can now be written as (starting at $t = 0$)

$$\begin{aligned} v^{\pi_0} &= c^{\pi_0} + \sum_{t'=1}^{\infty} \gamma^{t'} P^{\pi,t'} c_{t'}^{\pi} & (14.16) \\ &= c^{\pi_0} + \sum_{t'=1}^{\infty} \gamma^{t'} \left(\Pi_{t''=0}^{t'-1} P_{t''}^{\pi} \right) c_{t'}^{\pi} & (14.17) \\ &= c^{\pi_0} + \gamma P^{\pi_0} \sum_{t'=1}^{\infty} \gamma^{t'-1} \left(\Pi_{t''=1}^{t'-1} P_{t''}^{\pi} \right) c_{t'}^{\pi} & (14.18) \\ &= c^{\pi_0} + \gamma P^{\pi_0} v^{\pi}. & (14.19) \end{aligned}$$

Equation (14.19) 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 (14.19) as

$$v^{\pi} = c^{\pi} + \gamma P^{\pi} v^{\pi}. \qquad (14.20)$$

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

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

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 (14.11) would be written

$$\mathcal{M}^{\pi}(v) = c^{\pi} + \gamma P^{\pi} v. \qquad (14.21)$$

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.

14.4 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.

```
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:
        $v^n(s) = \max_{a \in \mathcal{A}} \big( C(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s,a) v^{n-1}(s') \big).$   (14.22)
Step 2. If $\|v^n - v^{n-1}\| < \epsilon(1-\gamma)/2\gamma$, let $\pi^\epsilon$ be the resulting policy that solves (14.22), and let $v^\epsilon = v^n$ and
        stop; else set $n = n + 1$ and go to step 1.
```

Figure 14.4 The value iteration algorithm for infinite horizon optimization

Value iteration comes in several flavors. The basic version of the value iteration algorithm is given in figure 14.4. The proof of convergence (see section 14.10.2) 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 ϵ 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.

14.4.1 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 14.5.

14.4.2 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

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 14.5 The Gauss-Seidel variation of value iteration.

Step 0. Initialization:

- Choose some $v^0 \in \mathcal{V}$.
- Choose a base state s^* and a tolerance ϵ.
- 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 $a^\epsilon = \arg\max_{a \in \mathcal{A}} \big(C(a) + \gamma P^\pi v^n \big)$.

Figure 14.6 Relative value iteration.

more quickly than the value function, which may grow steadily for many iterations. The relative value iteration algorithm is shown in 14.6.

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

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

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.

14.4.3 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 14.4.1. *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 14.10.3. 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

$$\|v^{n+1} - v^n\| < \epsilon(1-\gamma)/2\gamma \tag{14.23}$$

where γ is our discount factor and ϵ 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 14.4.2. *If we apply the value iteration algorithm with stopping parameter ϵ and the algorithm terminates at iteration n with value function v^{n+1}, then*

$$\|v^{n+1} - v^*\| \leq \epsilon/2. \tag{14.24}$$

Let π^ϵ be the policy that we terminate with, and let v^{π^ϵ} be the value of this policy. Then

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

The proof is given in section 14.10.4. 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

$$\begin{aligned} v^* &= \sum_{n=0}^{\infty} \gamma^n c \\ &= \frac{1}{1-\gamma} c. \end{aligned} \tag{14.25}$$

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

$$\begin{aligned} v^n &= \sum_{m=0}^{n-1} \gamma^n c \\ &= \frac{1-\gamma^n}{1-\gamma} c. \end{aligned} \tag{14.26}$$

Comparing equations (14.25) and (14.26), we see that

$$v^n - v^* = -\frac{\gamma^n}{1-\gamma} c. \tag{14.27}$$

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

$$\gamma^n c \leq \epsilon/2 \left(\frac{1-\gamma}{\gamma} \right). \tag{14.28}$$

If we choose ϵ so that (14.28) holds with equality, then our error bound (from 14.24) is

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

From (14.27), 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.

14.5 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 14.7. 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 14.3, the value of following policy π is given by

$$v^\pi \quad = \quad (I - \gamma P^\pi)^{-1} c^\pi. \tag{14.29}$$

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

$$A^\pi(R_t) = \begin{cases} 0, & R_t \geq q, \\ Q - R_t, & R_t < q. \end{cases} \tag{14.30}$$

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

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^π and contribution vector c^π, we can use equation (14.29) to find v^π, where $v^\pi(s)$ is the discounted value of started in state s and following policy π. From this vector, we can infer a new policy by solving

$$a^n(s) \quad = \quad \arg\max_{a \in \mathcal{A}} \big(C(a) + \gamma P^\pi v^n \big) \tag{14.31}$$

```
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
        $(I - \gamma P^{\pi^{n-1}})v = c^{\pi^{n-1}}.$
Step 3. Find a policy $\pi^n$ defined by
        $a^n(s) = \arg\max_{a \in \mathcal{A}} \big(C(a) + \gamma P^{\pi} v^n\big).$
    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 14.7 Policy iteration

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 (14.30). 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 (14.31). The complete policy iteration algorithm is described in figure 14.7.

The policy iteration algorithm is simple to implement and has fast convergence when measured in terms of the number of iterations. However, solving equation (14.29) 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.

14.6 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 14.8 outlines the procedure where the steady-state evaluation of the value function

```
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
    $$a^n(s) = \arg\max_{a \in \mathcal{A}} \left\{ C(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s,a) v^{n-1}(s') \right\},$$
    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 14.8 Hybrid value/policy iteration

in equation (14.29) is replaced with a much easier iterative procedure (step 2 in figure 14.8). 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.

14.7 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 π is given by

$$\max_\pi F^\pi(s) = \lim_{T \to \infty} \frac{1}{T} \mathbb{E} \sum_{t=0}^{T} C(S_t, A^\pi(S_t)). \tag{14.32}$$

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

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

where V_T^π 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

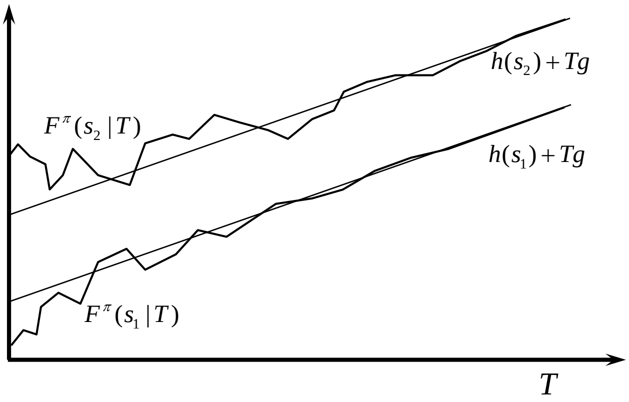


Figure 14.9 Cumulative contribution over a horizon T when starting in states s_1 and s_2, showing growth approaching a rate that is independent of the starting state.

is ergodic (all the states communicate with each other with positive probability), we know that $(P^\pi)^T \to P^*$ 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^π 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^π 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

$$V_T^\pi \to h^\pi + T g^\pi,$$

where h^π captures the state-dependent differences in the total contribution, while g^π is the state-independent average contribution in the limit. Figure 14.9 illustrates the growth in V_T^π toward a linear function.

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

14.8 THE LINEAR PROGRAMMING METHOD FOR DYNAMIC PROGRAMS

Theorem 14.4.1 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 β 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

$$\min_v \sum_{s \in \mathcal{S}} \beta_s v(s) \tag{14.33}$$

subject to

$$v(s) \quad \geq \quad C(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s,a)v(s') \quad \text{for all } s \text{ and } a, \tag{14.34}$$

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 (14.34)).

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.

14.9 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 14.1**

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 14.2**

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 14.3**

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.

14.9.1 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:

c^r = The fixed cost incurred each time we dispatch a new batch.

c^h = Penalty per time period for holding a unit of the resource.

K = Maximum size of a batch.

Our exogenous information process consists of

Q_t = Quantity of new arrivals during time interval t.

$\mathbb{P}^Q(i) = Prob(Q_t = i)$.

Our state variable is

R_t = Resources remaining at time t 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

$$
R_{t+1} = R_t - b_t + Q_{t+1}. \tag{14.35}
$$

The objective function is modeled using

$$
\begin{aligned}
C_t(R_t, a_t, b_t) &= \text{The cost incurred in period } t, \text{ given state } R_t \text{ and dispatch decision } a_t \\
&= c^r a_t + c^h(R_t - b_t). \qquad (14.36)
\end{aligned}
$$

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

$$
\min_{\pi \in \Pi} \mathbb{E} \left\{ \sum_{t=0}^{T} C_t(R_t, A_t^\pi(R_t)) \right\}. \tag{14.37}
$$

where Π 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 14 (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.

14.9.2 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 14.9.1. *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 14.9.2. *A function $g(u), u \in \mathcal{U}$ is* **supermodular** *if it satisfies*

$$g(u_1 \wedge u_2) + g(u_1 \vee u_2) \geq g(u_1) + g(u_2) \tag{14.38}$$

Supermodularity is the discrete analog of a convex function. A function is **submodular** if the inequality in equation (14.38) 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 14.9.3. *A function $g : \mathcal{U} \mapsto \Re^1$ is* **supermodular** *if it satisfies*

$$g(\mathcal{U}_1 \cup \mathcal{U}_2) + g(\mathcal{U}_1 \cap \mathcal{U}_2) \geq g(\mathcal{U}_1) + g(\mathcal{U}_2) \tag{14.39}$$

We may refer to definition 14.9.2 as the vector definition of supermodularity, while definition 14.9.3 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 14.9.1. *A function $g(s, a)$ is supermodular if for $s^+ \geq s^-$ and $a^+ \geq a^-$, then*

$$g(s^+, a^+) + g(s^-, a^-) \geq g(s^+, a^-) + g(s^-, a^+). \tag{14.40}$$

For our purposes, equation (14.40) will be the version we will use.

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

$$g(s^+, a^+) - g(s^-, a^+) \geq g(s^+, a^-) - g(s^-, a^-) \tag{14.41}$$

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

$$g(s^+, a^+) - g(s^+, a^-) \geq g(s^-, a^+) - g(s^-, a^-) \tag{14.42}$$

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 (14.40) 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 14.9.4. *A superadditive function $f : \Re^n \to \Re^1$ satisfies*

$$f(x) + f(y) \quad \leq \quad f(x + y). \tag{14.43}$$

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

14.9.3 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 14.9.1. *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 14.10.6.

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

$$C_t(R^+,1) - C_t(R^-,1) \leq C_t(R^+,0) - C_t(R^-,0). \tag{14.44}$$

Substituting equation (14.36) into (14.44), 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

$$(R^+ - K)^+ - (R^- - K)^+ \leq R^+ - R^-. \tag{14.45}$$

Since $R^+ \geq R^-$, $(R^+ - K)^+ = 0 \Rightarrow (R^- - K)^+ = 0$. This implies there are three possible cases for equation (14.45):

Case 1: $(R^+ - K)^+ > 0$ and $(R^- - K)^+ > 0$. In this case, (14.45) reduces to $R^+ - R^- = R^+ - R^-$.

Case 2: $(R^+ - K)^+ > 0$ and $(R^- - K)^+ = 0$. Here, (14.45) 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, (14.45) 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

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

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

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

which simplifies to

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

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

14.10 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 14.10.1 provides a proof that the optimal value function satisfies the optimality equations. Section 14.10.2 proves convergence of the value iteration algorithm. Section 14.10.3 then proves conditions under which value iteration increases or decreases monotonically to the optimal solution. Then, section 14.10.4 proves the bound on the error when value iteration satisfies the termination criterion given in section 14.4.3. Section 14.10.5 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.

14.10.1 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:

$$F_t^\pi(S_t) \quad = \quad \mathbb{E}\left\{\sum_{t'=t}^{T-1} C_{t'}(S_{t'}, A_{t'}^\pi(S_{t'})) + C_T(S_T)|S_t\right\}. \qquad (14.46)$$

Since (14.46) is, in general, exceptionally difficult to solve, we resort to the optimality equations

$$V_t^\pi(S_t) \quad = \quad C_t(S_t, A_t^\pi(S_t)) + \mathbb{E}\left\{V_{t+1}^\pi(S_{t+1})|S_t\right\}. \qquad (14.47)$$

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 14.10.1. *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*

$$\mathbb{E}\left[\mathbb{E}\{F_{t'}|S_{t+1}\}|S_t\right] \quad = \quad \mathbb{E}\left[F_{t'}|S_t\right]. \qquad (14.48)$$

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

$$\mathbb{E}\{F_{t'}|S_{t+1}\} \quad = \quad \sum_{f\in\mathcal{F}} f\mathbb{P}(F_{t'} = f|S_{t+1}). \qquad (14.49)$$

Recognizing that S_{t+1} is a random variable, we may take the expectation of both sides of (14.49), conditioned on S_t as follows:

$$\mathbb{E}\left[\mathbb{E}\{F_{t'}|S_{t+1}\}|S_t\right] = \sum_{S_{t+1}\in\mathcal{S}} \sum_{f\in\mathcal{F}} f\mathbb{P}(F_{t'} = f|S_{t+1}, S_t)\mathbb{P}(S_{t+1} = S_{t+1}|S_t). (14.50)$$

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 (14.50) (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 14.10.1. $F_t^\pi(S_t) = V_t^\pi(S_t)$.

Proof: To prove that (14.46) and (14.47) 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

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

We then use lemma 14.10.1 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 (14.47), we have a backward recursion for calculating $V_t^\pi(S_t)$ for a given policy π. Now that we can find the expected reward for a given π, we would like to find the best π. That is, we want to find

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

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

$$V_t(S_t) \quad = \quad \max_{a \in \mathcal{A}} \big(C_t(S_t, a) + \sum_{s' \in \mathcal{S}} p_t(s'|S_t, a) V_{t+1}(s') \big). \tag{14.51}$$

We are claiming that if we find the set of $V's$ that solves (14.51), then we have found the policy that optimizes F_t^π. We state this claim formally as:

Theorem 14.10.1. *Let $V_t(S_t)$ be a solution to equation (14.51). 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 π be an arbitrary policy. For $t' = t$, the optimality equation tells us

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

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

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

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

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

This means

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

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 π that satisfies

$$F_t^\pi(S_t) + (T-t)\epsilon \geq V_t(S_t). \tag{14.52}$$

To do this, we start with the definition

$$V_t(S_t) \quad = \quad \max_{a\in\mathcal{A}} \left(C_t(S_t,a) + \sum_{s'\in\mathcal{S}} p_t(s'|S_t,a)V_{t+1}(s') \right). \tag{14.53}$$

We may let $a_t(S_t)$ be the decision rule that solves (14.53). This rule corresponds to the policy π. 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

$$V_t(S_t) \quad \leq \quad C_t(S_t,a) + \sum_{s'\in\mathcal{S}} p_t(s'|S_t,a)V_{t+1}(s') + \epsilon. \tag{14.54}$$

We can prove (14.52) by induction. We first note that (14.52) 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

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

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

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

Now, using equation (14.54), we replace the term in brackets with the smaller $V_t(S_t)$ (equation (14.54)):

$$F^\pi_t(S_t) \quad \geq \quad V_t(S_t) - (T-t)\epsilon,$$

which proves the induction hypothesis. We have shown that

$$F^*_t(S_t) + (T-t)\epsilon \geq F^\pi_t(S_t) + (T-t)\epsilon \geq V_t(S_t) \geq F^*_t(S_t).$$

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.

14.10.2 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 14.10.1. *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 14.10.2. *$T : \mathcal{V} \rightarrow \mathcal{V}$ is a* **contraction mapping** *if there exists a γ, $0 \leq \gamma < 1$ such that:*

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

Definition 14.10.3. *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 14.10.4. *A normed linear space is* **complete** *if every Cauchy sequence contains a limit point in that space.*

Definition 14.10.5. *A* **Banach space** *is a complete normed linear space.*

Definition 14.10.6. *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 14.10.7. *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 14.10.2. *(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

$$\begin{aligned}
\|v^{n+m} - v^n\| &= \|\sum_{k=0}^{m-1}(v^{n+k+1} - v^{n+k})\| \\
&\le \sum_{k=0}^{m-1}\|(v^{n+k+1} - v^{n+k})\| \\
&= \sum_{k=0}^{m-1}\|(T^{n+k}v^1 - T^{n+k}v^0)\| \\
&\le \sum_{k=0}^{m-1}\gamma^{n+k}\|v^1 - v^0\| \\
&= \frac{\gamma^n(1-\gamma^m)}{(1-\gamma)}\|v^1 - v^0\|. \qquad (14.55)
\end{aligned}$$

Since $\gamma < 1$, for sufficiently large n the right-hand side of (14.55) 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

$$\lim_{n\to\infty} v^n \to v^*. \qquad (14.56)$$

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

$$\begin{aligned}
0 &\le \|Tv^* - v^*\| & (14.57)\\
&= \|Tv^* - v^n + v^n - v^*\| & (14.58)\\
&\le \|Tv^* - v^n\| + \|v^n - v^*\| & (14.59)\\
&= \|Tv^* - Tv^{n-1}\| + \|v^n - v^*\| & (14.60)\\
&\le \gamma\|v^* - v^{n-1}\| + \|v^n - v^*\|. & (14.61)
\end{aligned}$$

Equation (14.57) comes from the properties of a norm. We play our standard trick in (14.58) of adding and subtracting a quantity (in this case, v^n), which sets up the triangle inequality in (14.59). Using $v^n = Tv^{n-1}$ gives us (14.60). The inequality in (14.61) is based on the assumption of the theorem that T is a contraction mapping. From (14.56), we know that

$$\lim_{n\to\infty}\|v^* - v^{n-1}\| = \lim_{n\to\infty}\|v^n - v^*\| = 0. \qquad (14.62)$$

Combining (14.57), (14.61), and (14.62) gives

$$0 \le \|Tv^* - v^*\| \le 0$$

from which we conclude

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

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

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

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 14.10.2. *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 elementwise. 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

$$\begin{aligned}
0 &\leq \mathcal{M}v(s) - \mathcal{M}u(s) && (14.63)\\
&= C(s, a_s^*(v)) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a_s^*(v))v(s') \\
&\quad - \left(C(s, a_s^*(u)) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a_s^*(u))u(s') \right) && (14.64)\\
&\leq C(s, a_s^*(v)) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a_s^*(v))v(s') \\
&\quad - \left(C(s, a_s^*(v)) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a_s^*(v))u(s') \right) && (14.65)\\
&= \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a_s^*(v))[v(s') - u(s')] && (14.66)\\
&\leq \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a_s^*(v))\|v - u\| && (14.67)\\
&= \gamma \|v - u\| \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a_s^*(v)) && (14.68)\\
&= \gamma \|v - u\|. && (14.69)
\end{aligned}$$

Equation (14.63) is true by assumption, while (14.64) holds by definition. The inequality in (14.65) 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 (14.66) is a simple reduction of (14.65). Equation (14.67) 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 (14.68), which then easily reduces to (14.69) 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

$$|\mathcal{M}v(s) - \mathcal{M}u(s)| \leq \gamma |v(s) - u(s)| \qquad (14.70)$$

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. □

14.10.3 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 14.10.3. *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

$$\begin{aligned} v &\geq \max_{\pi\in\Pi}\{c^\pi + \gamma P^\pi v\} && (14.71)\\ &\geq c^{\pi_0} + \gamma P^{\pi_0} v && (14.72)\\ &\geq c^{\pi_0} + \gamma P^{\pi_0}\left(c^{\pi_1} + \gamma P^{\pi_1} v\right) && (14.73)\\ &= c^{\pi_0} + \gamma P^{\pi_0} c^{\pi_1} + \gamma^2 P^{\pi_0} P^{\pi_1} v. \end{aligned}$$

Equation (14.71) is true by assumption (part (a) of the theorem) and equation (14.72) is true because π_0 is some policy that is not necessarily optimal for the vector v. Using similar reasoning, equation (14.73) is true because π_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

$$v \geq c^{\pi_0} + \gamma P^{\pi_0} c^{\pi_1} + \cdots + \gamma^{t-1} P^{\pi_0} P^{\pi_1} \cdots P^{\pi_{t-1}} c^{\pi_t} + \gamma^t P^{\pi,(t)} v. \quad (14.74)$$

Recall that

$$v^\pi = \sum_{t=0}^{\infty} \gamma^t P^{\pi,(t)} c^{\pi_t}. \quad (14.75)$$

Breaking the sum in (14.75) into two parts allows us to rewrite the expansion in (14.74) as

$$v \quad \geq \quad v^\pi - \sum_{t'=t+1}^{\infty} \gamma^{t'} P^{\pi,(t')} c^{\pi_{t'+1}} + \gamma^t P^{\pi,(t)} v. \tag{14.76}$$

Taking the limit of both sides of (14.76) as $t \to \infty$ gives us

$$\begin{aligned} v \quad &\geq \quad \lim_{t\to\infty} v^\pi - \sum_{t'=t+1}^{\infty} \gamma^{t'} P^{\pi,(t')} c^{\pi_{t'+1}} + \gamma^t P^{\pi,(t)} v && (14.77) \\ &\geq \quad v^\pi \quad \forall \pi \in \Pi. && (14.78) \end{aligned}$$

The limit in (14.77) exists as long as the reward function c^π is bounded and $\gamma < 1$. Because (14.78) is true for all $\pi \in \Pi$, it is also true for the optimal policy, which means that

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

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.

14.10.4 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 π^ϵ be the policy that satisfies our stopping rule, and let v^{π^ϵ} be the infinite horizon value of following policy π^ϵ.

Theorem 14.10.4. *If we apply the value iteration algorithm with stopping parameter ϵ and the algorithm terminates at iteration n with value function v^{n+1}, then*

$$\|v^{n+1} - v^*\| \leq \epsilon/2, \tag{14.79}$$

and

$$\|v^{\pi^\epsilon} - v^*\| \leq \epsilon. \tag{14.80}$$

Proof: We start by writing

$$\begin{aligned} \|v^{\pi^\epsilon} - v^*\| \quad &= \quad \|v^{\pi^\epsilon} - v^{n+1} + v^{n+1} - v^*\| \\ &\leq \quad \|v^{\pi^\epsilon} - v^{n+1}\| + \|v^{n+1} - v^*\|. && (14.81) \end{aligned}$$

Recall that π^ϵ 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 (14.81) 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 (14.81) to show that

$$\|v^{n+1} - v^*\| \quad \leq \quad \frac{\gamma}{1-\gamma}\|v^{n+1} - v^{n}\|. \tag{14.82}$$

The value iteration algorithm stops when $\|v^{n+1} - v^n\| \leq \epsilon(1-\gamma)/2\gamma$. Substituting this in (14.82) gives

$$\|v^{n+1} - v^*\| \quad \leq \quad \frac{\epsilon}{2}. \tag{14.83}$$

Recognizing that the same bound applies to $\|v^{\pi^\epsilon} - v^{n+1}\|$ and combining these with (14.81) gives us

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

which completes our proof. □

14.10.5 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 π (more precisely, decision rule A^π).

In this case, our optimality equations look like

$$V_t^*(S_t) = \max_{\pi\in\Pi^{MR}} \sum_{a\in\mathcal{A}} \left[q_t^\pi(a|S_t) \left(C_t(S_t,a) + \sum_{s'\in\mathcal{S}} p_t(s'|S_t,a)V_{t+1}^*(s') \right) \right]. \tag{14.84}$$

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

$$C_t(S_t,a^*) + \sum_{s'\in\mathcal{S}} p_t(s'|S_t,a^*)V_{t+1}^*(s') \quad \geq \quad C_t(S_t,a) + \sum_{s'\in\mathcal{S}} p_t(s'|S_t,a)V_{t+1}^*(s') \tag{14.85}$$

for all $a \in \mathcal{A}$. Substituting (14.85) back into (14.84) 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.

14.10.6 Optimality of monotone policies

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

Lemma 14.10.2. *If a function $g(s,a)$ is supermodular, then*

$$a^*(s) \quad = \quad \max\left\{ a' \in \arg\max_a g(s,a) \right\} \tag{14.86}$$

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 (14.86) with

$$a^*(s) \quad = \quad \max_a g(s,a). \tag{14.87}$$

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 (14.42), 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

$$g(s^-, a^*(s^-)) - g(s^-, a) \quad \geq \quad 0. \tag{14.88}$$

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

$$g(s^-, a) + g(s^+, a^*(s^-)) \quad \geq \quad g(s^-, a^*(s^-)) + g(s^+, a) \tag{14.89}$$

Rearranging (14.89) gives us

$$g(s^+, a^*(s^-)) \geq \underbrace{\{g(s^-, a^*(s^-)) - g(s^-, a)\}}_{\geq 0} + g(s^+, a) \quad \forall a \leq a^*(s^-) \tag{14.90}$$

$$\geq \quad g(s^+, a) \qquad \forall a \leq a^*(s^-) \tag{14.91}$$

We obtain equation (14.91) because the term in brackets in (14.90) is nonnegative (from (14.88)).

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 14.10.3. *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 14.10.5. *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

$$w(s,a) \quad = \quad C_t(s,a) + \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s,a) v_{t+1}(s') \tag{14.92}$$

The two terms on the right-hand side of (14.92) 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 14.10.2, 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 14.10.3. *Let $p_j, p'_j, j \in \mathcal{J}$ be probability mass functions defined over $\mathcal{J}$ that satisfy*

$$\sum_{j=j'}^{\infty} p_j \quad \geq \quad \sum_{j=j'}^{\infty} p'_j \quad \forall j' \in \mathcal{J} \tag{14.93}$$

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

$$\sum_{j=0}^{\infty} p_j v_j \quad \geq \quad \sum_{j=0}^{\infty} p'_j v_j \tag{14.94}$$

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 (14.94) 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

$$\begin{aligned}
\sum_{j=0}^{\infty} p_j v_j \quad &= \quad \sum_{j=0}^{\infty} p_j \sum_{i=0}^{j} (v_i - v_{i-1}) && (14.95)\\
&= \quad \sum_{j=0}^{\infty} (v_j - v_{j-1}) \sum_{i=j}^{\infty} p_i && (14.96)\\
&= \quad \sum_{j=1}^{\infty} (v_j - v_{j-1}) \sum_{j=i}^{\infty} p_i + v_0 \sum_{i=0}^{\infty} p_i && (14.97)\\
&\geq \quad \sum_{j=1}^{\infty} (v_j - v_{j-1}) \sum_{i=j}^{\infty} p'_j + v_0 \sum_{i=0}^{\infty} p'_j && (14.98)\\
&= \quad \sum_{j=0}^{\infty} p'_j v_j && (14.99)
\end{aligned}$$

In equation (14.95), we replace v_j with an alternating sequence that sums to v_j. Equation (14.96) involves one of those painful change of variable tricks with summations. Equation (14.97) is simply getting rid of the term that involves v_{-1}. In equation (14.98), 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 (14.99).□

We stated that lemma 14.10.3 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 (14.94) 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 14.10.4. *Lemma 14.10.3 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 14.10.3 never required that the sequences $\{p_j\}$ and $\{p'_j\}$ be probability mass functions.

Proposition 14.10.4. *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:

$$\begin{aligned} v_t(s) &= \max_{a \in \mathcal{A}} C_t(s, a) + \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a) v_{t+1}(s') \\ &= C_t(s, a^*_t(s)) + \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a^*_t(s)) v_{t+1}(s') \end{aligned} \tag{14.100}$$

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

$$\sum_{s' \geq s} \mathbb{P}(s'|s, a) \leq \sum_{s' \geq s} \mathbb{P}(s'|\hat{s}, a) \tag{14.101}$$

Lemma 14.10.4 tells us that when (14.101) holds, and if $v_{t+1}(s')$ is nondecreasing (the induction hypothesis), then:

$$\sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a) v_{t+1}(s') \leq \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|\hat{s}, a) v_{t+1}(s') \tag{14.102}$$

Combining equation (14.102) with condition (a) of proposition 14.10.4 into equation (14.100) 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 14.10.5:

Proposition 14.10.5. *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 14.10.4 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.

14.11 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), **?**, 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 **?**. Most of this chapter is based on **?**, modified to our notational style.

Section 14.8 - The linear programming method was first proposed in Manne (1960) (see subsequent discussions in **?** and **?**). 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 18.8.

Section 14.10.6 - In addition to **?**, see also Topkins (1978).

PROBLEMS

14.1 Discrete Markov decision processes have been studied since the 1950's as a way of solving stochastic, dynamic programs. Yet, in chapter 4, this is used as an example of a stochastic optimization problem that can be solved deterministically. Explain.

14.2 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 $p^\pi_{ij} = \mathbb{P}(R_{t+1} = j | R_t = i, A^\pi = a_t)$.

14.3 Repeat the previous exercise, but now assume that we have adopted a policy π 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 π (which describes both the structure of the policy and the control parameter s).

14.4 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.

14.5 Show that $\mathbb{P}(S_{t+\tau}|S_t)$, given that we are following a policy π (for stationary problems), is given by (14.14). [Hint: first show it for $\tau = 1, 2$ and then use inductive reasoning to show that it is true for general τ.]

14.6 Apply policy iteration to the problem given in exercise 14.4. 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.

14.7 Now apply the hybrid value-policy iteration algorithm to the problem given in exercise 14.4. 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.

14.8 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.

14.9 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.

14.10 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.

14.11 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.

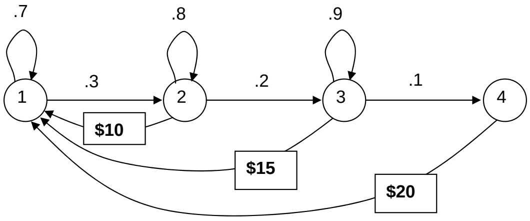
(b) Give the expected reward function for each time period and the expected terminal reward function.

(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.



14.12 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.

14.13 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.

	Iteration				
State	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

14.14 In the proof of theorem 14.10.3 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 $v \leq v^*$.

14.15 Theorem 14.10.3 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 $m \geq n$.

14.16 Consider a finite-horizon MDP with the following properties:

- $\mathcal{S} \in \Re^n$, 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 ω_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^*.

14.17 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 ε_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 τ 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.

14.18 Show that the matrix H^n in the recursive updating formula from equation (3.81)

$$\bar{\theta}^n = \bar{\theta}^{n-1} - H^n x^n \hat{\varepsilon}^n$$

reduces to $H^n = 1/n$ for the case of a single parameter (which means we are using Y =constant, with no independent variables).

14.19 An airline has to decide when to bring an aircraft in for a major engine overhaul. Let s_t represent the state of the engine in terms of engine wear, and let d_t be a nonnegative amount by which the engine deteriorates during period t. At the beginning of period t, the airline may decide to continue operating the aircraft ($z_t = 0$) or to repair the aircraft ($z_t = 1$) at a cost of c^R, which has the effect of returning the aircraft to $s_{t+1} = 0$. If the airline does not repair the aircraft, the cost of operation is $c^o(s_t)$, which is a nondecreasing, convex function in s_t.

(a) Define what is meant by a control limit policy in dynamic programming, and show that this is an instance of a monotone policy.

(b) Formulate the one-period reward function $C_t(s_t, z_t)$, and show that it is submodular.

(c) Show that the decision rule is monotone in s_t. (Outline the steps in the proof, and then fill in the details.)

(d) Assume that a control limit policy exists for this problem, and let γ be the control limit. Now, we may write $C_t(s_t, z_t)$ as a function of one variable: the state s. Using our control limit structure, we can write the decision z_t as the decision rule $z^\pi(s_t)$. Illustrate the shape of $C_t(s_t, z^\pi(s))$ by plotting it over the range $0 \leq s \leq 3\gamma$ (in theory, we may be given an aircraft with $s > \gamma$ initially).

14.20 A dispatcher controls a finite capacity shuttle that works as follows: In each time period, a random number A_t arrives. After the arrivals occur, the dispatcher must decide whether to call the shuttle to remove up to M customers. The cost of dispatching the shuttle is c, which is independent of the number of customers on the shuttle. Each time period that a customer waits costs h. If we let $z = 1$ if the shuttle departs and 0 otherwise, then our one-period reward function is given by

$$c_t(s, z) = cz + h[s - Mz]^+,$$

where M is the capacity of the shuttle. Show that $c_t(s, a)$ is submodular where we would like to minimize r. Note that we are representing the state of the system after the customers arrive.

14.21 Assume that a control limit policy exists for our shuttle problem in exercise 2 that allows us to write the optimal dispatch rule as a function of s, as in $z^\pi(s)$. We may write $r(s, z)$ as a function of one variable, the state s.

(a) Illustrate the shape of $r(s, z(s))$ by plotting it over the range $0 < s < 3M$ (since we are allowing there to be more customers than can fill one vehicle, assume that we are allowed to send $z = 0, 1, 2, \ldots$ vehicles in a single time period).

(b) Let $c = 10$, $h = 2$, and $M = 5$, and assume that $A_t = 1$ with probability 0.6 and is 0 with probability 0.4. Set up and solve a system of linear equations for the optimal value function for this problem in steady state.

14.22 A general aging and replenishment problem arises as follows: Let s_t be the "age" of our process at time t. At time t, we may choose between a decision $d = C$ to continue the process, incurring a cost $g(s_t)$ or a decision $d = R$ to replenish the process, which incurs a cost $K + g(0)$. Assume that $g(s_t)$ is convex and increasing. The state of the system evolves according to

$$s_{t+1} = \begin{cases} s_t + D_t & \text{if } d = C, \\ 0 & \text{if } d = R, \end{cases}$$

where D_t is a nonnegative random variable giving the degree of deterioration from one epoch to another (also called the "drift").

(a) Prove that the structure of this policy is monotone. Clearly state the conditions necessary for your proof.

(b) How does your answer to part (1) change if the random variable D_t is allowed to take on negative outcomes? Give the weakest possible conditions on the distribution of required to ensure the existence of a monotone policy.

(c) Now assume that the action is to reduce the state variable by an amount $q \leq s_t$ at a cost of cq (instead of K). Further assume that $g(s) = as^2$. Show that this policy is also monotone. Say as much as you can about the specific structure of this policy.

14.23 Show that the matrix H^n in the recursive updating formula from equation (3.81)

$$\bar{\theta}^n = \bar{\theta}^{n-1} - H^n x^n \hat{\varepsilon}^n$$

reduces to $H^n = 1/n$ for the case of a single parameter (which means we are using Y =constant, with no independent variables).

14.24 An airline has to decide when to bring an aircraft in for a major engine overhaul. Let s_t represent the state of the engine in terms of engine wear, and let d_t be a nonnegative amount by which the engine deteriorates during period t. At the beginning of period t, the airline may decide to continue operating the aircraft ($z_t = 0$) or to repair the aircraft ($z_t = 1$) at a cost of c^R, which has the effect of returning the aircraft to $s_{t+1} = 0$. If the airline does not repair the aircraft, the cost of operation is $c^o(s_t)$, which is a nondecreasing, convex function in s_t.

(a) Define what is meant by a control limit policy in dynamic programming, and show that this is an instance of a monotone policy.

(b) Formulate the one-period reward function $C_t(s_t, z_t)$, and show that it is submodular.

(c) Show that the decision rule is monotone in s_t. (Outline the steps in the proof, and then fill in the details.)

(d) Assume that a control limit policy exists for this problem, and let γ be the control limit. Now, we may write $C_t(s_t, z_t)$ as a function of one variable: the state s. Using our control limit structure in section 14.9.3, we can write the decision z_t as the decision rule $z^\pi(s_t)$. Illustrate the shape of $C_t(s_t, z^\pi(s))$ by plotting it over the range $0 \leq s \leq 3\gamma$ (in theory, we may be given an aircraft with $s > \gamma$ initially).

14.25 A dispatcher controls a finite capacity shuttle that works as follows: In each time period, a random number A_t arrives. After the arrivals occur, the dispatcher must decide whether to call the shuttle to remove up to M customers. The cost of dispatching the shuttle is c, which is independent of the number of customers on the shuttle. Each time period that a customer waits costs h. If we let $z = 1$ if the shuttle departs and 0 otherwise, then our one-period reward function is given by

$$c_t(s, z) = cz + h[s - Mz]^+,$$

where M is the capacity of the shuttle. Show that $c_t(s, a)$ is submodular where we would like to minimize r. Note that we are representing the state of the system after the customers arrive.

14.26 Assume that a control limit policy exists for our shuttle problem in exercise 2 that allows us to write the optimal dispatch rule as a function of s, as in $z^\pi(s)$. We may write $r(s, z)$ as a function of one variable, the state s.

(a) Illustrate the shape of $r(s, z(s))$ by plotting it over the range $0 < s < 3M$ (since we are allowing there to be more customers than can fill one vehicle, assume that we are allowed to send $z = 0, 1, 2, \ldots$ vehicles in a single time period).

(b) Let $c = 10$, $h = 2$, and $M = 5$, and assume that $A_t = 1$ with probability 0.6 and is 0 with probability 0.4. Set up and solve a system of linear equations for the optimal value function for this problem in steady state.

14.27 A general aging and replenishment problem arises as follows: Let s_t be the "age" of our process at time t. At time t, we may choose between a decision $d = C$ to continue the process, incurring a cost $g(s_t)$ or a decision $d = R$ to replenish the process, which incurs a cost $K + g(0)$. Assume that $g(s_t)$ is convex and increasing. The state of the system evolves according to

$$s_{t+1} = \begin{cases} s_t + D_t & \text{if } d = C, \\ 0 & \text{if } d = R, \end{cases}$$

where D_t is a nonnegative random variable giving the degree of deterioration from one epoch to another (also called the "drift").

(a) Prove that the structure of this policy is monotone. Clearly state the conditions necessary for your proof.

(b) How does your answer to part (1) change if the random variable D_t is allowed to take on negative outcomes? Give the weakest possible conditions on the distribution of required to ensure the existence of a monotone policy.

(c) Now assume that the action is to reduce the state variable by an amount $q \leq s_t$ at a cost of cq (instead of K). Further assume that $g(s) = as^2$. Show that this policy is also monotone. Say as much as you can about the specific structure of this policy.

CHAPTER 15

DYNAMIC PROGRAMS WITH SPECIAL STRUCTURE

15.1 OPTIMAL MYOPIC POLICIES

15.2 SPECIAL CASES WITH ANALYTICAL SOLUTIONS

Square/square-root/log cost functions.

15.3 SIMPLIFIED POST-DECISION STATE VARIABLE

Produces optimal myopic policies.

15.3.1 Empty post-decision state variable

15.3.2 Discrete post-decision state variable

Example on graphs - small number of discrete states. Use JOC article.
Shortest path where costs are revealed when you arrive at a node.

15.4 MONOTONE DYNAMIC PROGRAMMING

Move material from MDP chapter.

15.5 LINEAR-QUADRATIC REGULATION

15.5.1 Budgeting problems

The budgeting problem is a simple resource allocation problem where we start with a resource R which then has to be allocated over a series of activities.

15.5.1.1 The discrete budgeting problem Assume we have to allocate a budget of size R to a series of tasks $\mathcal{T}$. Let x_t be a discrete action representing the amount of money allocated to task t, and let $C_t(x_t)$ be the contribution (or reward) that we receive from this allocation. We would like to maximize our total contribution

$$\max_x \sum_{t\in\mathcal{T}} C_t(x_t) \tag{15.1}$$

subject to the constraint on our available resources

$$\sum_{t\in\mathcal{T}} x_t = R. \tag{15.2}$$

In addition, we cannot allocate negative resources to any task, so we include

$$x_t \geq 0. \tag{15.3}$$

We refer to (15.1)-(15.3) as the *budgeting problem* (other authors refer to it as the "resource allocation problem," a term we find too general for such a simple problem). In this example, all data are deterministic. There are a number of algorithmic strategies for solving this problem that depend on the structure of the contribution function, but we are going to show how it can be solved without any assumptions.

We will approach this problem by first deciding how much to allocate to task 1, then to task 2, and so on, until the last task, T. In the end, however, we want a solution that optimizes over all tasks. Let

$V_t(R_t) =$ The value of having R_t resources remaining to allocate to task t and later tasks.

Implicit in our definition of $V_t(R_t)$ is that we are going to solve the problem of allocating R_t over tasks $t, t+1, \ldots, T$ in an optimal way. Imagine that we somehow know the function $V_{t+1}(R_{t+1})$. The relationship between R_{t+1} and R_t is given by

$$R_{t+1} = R_t - x_t. \tag{15.4}$$

In the language of dynamic programming, R_t is known as the *state variable*, which captures all the information we need to model the system forward in time (we provide a more careful definition in chapter 9). Equation (15.4) is the *transition function* which relates the state at time t to the state at time $t+1$. Sometimes we need to explicitly refer to the transition function (rather than just the state at time $t+1$), in which case we use

$$R^M(R_t, x_t) = R_t - x_t. \tag{15.5}$$

Equation (15.5) is referred to in some communities as the *system model*, since it models the physics of the system over time (hence our use of the superscript M).

The relationship between $V_t(R_t)$ and $V_{t+1}(R_{t+1})$ is given by

$$V_t(R_t) = \max_{0 \le x_t \le R_t} \left(C_t(x_t) + V_{t+1}(R^M(R_t, x_t)) \right). \tag{15.6}$$

Equation (15.6) is the optimality equation and represents the foundational equation for dynamic programming. It says that the value of having R_t resources for task t is the value of optimizing the contribution from task t plus the value of then having $R^M(R_t, x_t) = R_{t+1} = R_t - x_t$ resources for task $t+1$ (and beyond). It forces us to balance the contribution from task t against the value that we would receive from all future tasks (which is captured in $V_{t+1}(R_t - x_t)$). One way to solve (15.6) is to assume that x_t is discrete. For example, if our budget is $R =$ \$10 million, we might require x_t to be in units of \$100,000 dollars. In this case, we would solve (15.6) simply by searching over all possible values of x_t (since it is a scalar, this is not too hard). The problem is that we do not know what $V_{t+1}(R_{t+1})$ is.

The simplest strategy for solving our dynamic program in (15.6) is to start by using $V_{T+1}(R) = 0$ (for any value of R). Then we would solve

$$V_T(R_T) = \max_{0 \le x_T \le R_T} C_T(x_T) \tag{15.7}$$

for $0 \le R_T \le R$. Now we know $V_T(R_T)$ for any value of R_T that might actually happen. Next we can solve

$$V_{T-1}(R_{T-1}) = \max_{0 \le x_{T-1} \le R_{T-1}} \left(C_{T-1}(x_{T-1}) + V_T(R_{T-1} - x_{T-1}) \right). \tag{15.8}$$

Clearly, we can play this game recursively, solving (15.6) for $t = T-1, T-2, \ldots, 1$. Once we have computed V_t for $t = (1, 2, \ldots, T)$, we can then start at $t = 1$ and step forward in time to determine our optimal allocations.

This strategy is simple, easy, and optimal. It has the nice property that we do not need to make any assumptions about the shape of $C_t(x_t)$, other than finiteness. We do not need concavity or even continuity; we just need the function to be defined for the discrete values of x_t that we are examining.

15.5.1.2 The continuous budgeting problem It is usually the case that dynamic programs have to be solved numerically. In this section, we introduce a form of the budgeting problem that can be solved analytically. Assume that the resources we are allocating are continuous (for example, how much money to assign to various activities), which means that R_t is continuous, as is the decision of how much to budget. We are going to assume that the contribution from allocating x_t dollars to task t is given by

$$C_t(x_t) = \sqrt{x_t}.$$

This function assumes that there are diminishing returns from allocating additional resources to a task, as is common in many applications. We can solve this problem exactly using dynamic programming. We first note that if we have R_T dollars left for the last task, the value of being in this state is

$$V_T(R_T) = \max_{x_T \le R_T} \sqrt{x_T}.$$

Since the contribution increases monotonically with x_T, the optimal solution is $x_T = R_T$, which means that $V_T(R_T) = \sqrt{R_T}$. Now consider the problem at time $t = T - 1$. The value of being in state R_{T-1} would be

$$V_{T-1}(R_{T-1}) = \max_{x_{T-1} \le R_{T-1}} \left(\sqrt{x_{T-1}} + V_T(R_T(x_{T-1})) \right) \tag{15.9}$$

where $R_T(x_{T-1}) = R_{T-1} - x_{T-1}$ is the money left over from time period $T-1$. Since we know $V_T(R_T)$ we can rewrite (15.9) as

$$V_{T-1}(R_{T-1}) = \max_{x_{T-1} \leq R_{T-1}} \left(\sqrt{x_{T-1}} + \sqrt{R_{T-1} - x_{T-1}}\right). \tag{15.10}$$

We solve (15.10) by differentiating with respect to x_{T-1} and setting the derivative equal to zero (we are taking advantage of the fact that we are maximizing a continuously differentiable, concave function). Let

$$F_{T-1}(R_{T-1}, x_{T-1}) = \sqrt{x_{T-1}} + \sqrt{R_{T-1} - x_{T-1}}.$$

Differentiating $F_{T-1}(R_{T-1}, x_{T-1})$ and setting this equal to zero gives

$$\begin{aligned} \frac{\partial F_{T-1}(R_{T-1}, x_{T-1})}{\partial x_{T-1}} &= \frac{1}{2}(x_{T-1})^{-\frac{1}{2}} - \frac{1}{2}(R_{T-1} - x_{T-1})^{-\frac{1}{2}} \\ &= 0. \end{aligned}$$

This implies

$$x_{T-1} = R_{T-1} - x_{T-1}$$

which gives

$$x^*_{T-1} = \frac{1}{2} R_{T-1}.$$

We now have to find V_{T-1}. Substituting x^*_{T-1} back into (15.10) gives

$$\begin{aligned} V_{T-1}(R_{T-1}) &= \sqrt{R_{T-1}/2} + \sqrt{R_{T-1}/2} \\ &= 2\sqrt{R_{T-1}/2}. \end{aligned}$$

We can continue this exercise, but there seems to be a bit of a pattern forming (this is a common trick when trying to solve dynamic programs analytically). It seems that a general formula might be

$$V_{T-t+1}(R_{T-t+1}) = t\sqrt{R_{T-t+1}/t} \tag{15.11}$$

or, equivalently,

$$V_t(R_t) = (T-t+1)\sqrt{R_t/(T-t+1)}. \tag{15.12}$$

How do we determine if this guess is correct? We use a technique known as proof by induction. We assume that (15.11) is true for $V_{T-t+1}(R_{T-t+1})$ and then show that we get the same structure for $V_{T-t}(R_{T-t})$. Since we have already shown that it is true for V_T and V_{T-1}, this result would allow us to show that it is true for all t.

Finally, we can determine the optimal solution using the value function in equation (15.12). The optimal value of x_t is found by solving

$$\max_{x_t} \left(\sqrt{x_t} + (T-t)\sqrt{(R_t - x_t)/(T-t)}\right). \tag{15.13}$$

Differentiating and setting the result equal to zero gives

$$\frac{1}{2}(x_t)^{-\frac{1}{2}} - \frac{1}{2}\left(\frac{R_t - x_t}{T-t}\right)^{-\frac{1}{2}} = 0.$$

This implies that

$$x_t = (R_t - x_t)/(T - t).$$

Solving for x_t gives

$$x_t^* = R_t/(T - t + 1).$$

This gives us the very intuitive result that we want to evenly divide the available budget among all remaining tasks. This is what we would expect since all the tasks produce the same contribution.

15.6 BIBLIOGRAPHIC NOTES

- Section xx -

PROBLEMS

15.1 Repeat the derivation in section 15.5.1.2 assuming that the reward for task t is $c_t\sqrt{x_t}$.

15.2 Repeat the derivation in section 15.5.1.2 assuming that the reward for task t is given by $\ln(x)$.

15.3 Repeat the derivation in section 15.5.1.2 one more time, but now assume that all you know is that the reward is continuously differentiable, monotonically increasing and concave.

15.4 What happens to the answer to the budget allocation problem in section 15.5.1.2 if the contribution is convex instead of concave (for example, $C_t(x_t) = x_t^2$)?

15.5 You have to send a set of questionnaires to each of N population segments. The size of each population segment is given by w_i. You have a budget of B questionnaires to allocate among the population segments. If you send x_i questionnaires to segment i, you will have a sampling error proportional to

$$f(x_i) = 1/\sqrt{x_i}.$$

You want to minimize the weighted sum of sampling errors, given by

$$F(x) = \sum_{i=1}^{N} w_i f(x_i)$$

You wish to find the allocation x that minimizes $F(x)$ subject to the budget constraint $\sum_{i=1}^{N} x_i \leq B$. Set up the optimality equations to solve this problem as a dynamic program (needless to say, we are only interested in integer solutions).

CHAPTER 16

BACKWARD APPROXIMATE DYNAMIC PROGRAMMING

Chapter 14 presented the most classical solution methods from discrete Markov decision processes, which are often referred to as "backward dynamic programming" since it is necessary to step backward in time, using the value $V_{t+1}(S_{t+1})$ to compute $V_t(S_t)$. While we can occasionally apply this strategy to problems with continuous states and actions (as we showed in chapter 15), most often this is used for problems with discrete states and actions, and where the one-step transition matrix $P(S_{t+1} = s'|S_t = s, a)$.

The basic backward dynamic programming strategy summarized in figure 14.3 suffers from what we have identified as the three curses of dimensionality:

1) State variables - As the state variable grows past two or three dimensions, the number of states tends to become too large to enumerate.

2) Action (decision) variables - Enumerating all possible actions tends to become intractable if there are more than three or four dimensions, unless it is possible to significantly prune the number of actions using constraints. Problems with more than five or six dimensions tend to require special structure such as convexity.

3) Exogenous information - As we pointed out in section 9.6 finding the one-step transition matrix requires computing the expectation

$$\begin{aligned} P(s'|s,x) &= \mathbb{E}\{\mathbb{1}_{\{s'=S^M(S_t,x,W_{t+1})\}}|S_t = s\} \\ &= \sum_{\omega_{t+1}\in\Omega_{t+1}} P(W_{t+1} = \omega_{t+1}) 1_{\{s'=S^M(S_t,x,\omega_{t+1})\}}. \end{aligned} \tag{16.1}$$

If W_{t+1} is a vector (again, with more than two or three dimensions), this becomes computationally intractable.

This is best illustrated by rewriting the algorithm in figure 14.3 for a problem where the state $s = (s_1, s_2, s_3)$, action $a = (a_1, a_2, a_3)$ and exogenous information $w = (w_1, w_2, w_3)$ each have three dimensions. The resulting algorithm is summarized in figure 16.1, where we have oversimplified the challenges by not showing the nesting of each of the dimensions. However, as written, there are 10 nested loops: one going backward in time, three for the states, three for the actions, and three expressed as nested sums for the exogenous information. Depending on how many values there are for each state, action and exogenous information, this simple recursion could easily require a year or more to execute.

```
**Step 0.** Initialization:
    **Step 0a.** Initialize the terminal contribution $V_T(S_T)$.
    **Step 0b.** Set $t = T - 1$.
**Step 1a.** Step backward in time $t = T, T-1, \ldots, 0$:
    **Step 2a.** Loop over states $s_1 \in \mathcal{S}_1$:
    **Step 2b.** Loop over states $s_2 \in \mathcal{S}_2$:
    **Step 2c.** Loop over states $s_3 \in \mathcal{S}_3$:
    **Step 2d.** Let $s = (s_1, s_2, s_3)$.
    **Step 2e.** Initialize $V_t(s) = -M$ (where $M$ is very large).
        **Step 3a.** Loop over each action $a_1 \in \mathcal{A}_1(s)$:
        **Step 3b.** Loop over each action $a_2 \in \mathcal{A}_2(s)$:
        **Step 3c.** Loop over each action $a_3 \in \mathcal{A}_3(s)$:
        **Step 3d.** Let $a = (a_1, a_2, a_3)$.
            **Step 4a.** Initialize $Q(s, a) = 0$.
            **Step 4b.** Find the expected value of being in state $s$ and taking action $a$:
            **Step 4c.** Compute $Q_t(s, a) =$
  $\sum_{w_1 \in \mathcal{W}_1} \sum_{w_2 \in \mathcal{W}_2} \sum_{w_3 \in \mathcal{W}_3} \mathbb{P}(w_1, w_2, w_3 | s, a) V_{t+1}(s' = s^M(s, a, (w_1, w_2, w_3)))$.
        **Step 3e.** If $Q_t(s, a) > V_t(s)$ then
            **Step 3f.** Store the best value $V_t(s) = Q_t(s, a)$.
            **Step 3g.** Store the best action $A_t(s) = a$.
**Step 1b.** Return the value $V_t(s)$ and policy $A_t(s)$ for all $s \in \mathcal{S}$ and $t = 0, \ldots, T$.
```

Figure 16.1 A backward dynamic programming algorithm.

These computational issues have motivated the development of fields with names like "approximate dynamic programming," "adaptive dynamic programming," (the term more widely used in engineering), "neuro-dynamic programming," or "reinforcement learning," (the highly popular field that evolved within computer science). In this book, we refer to all of these approaches as "forward approximate dynamic programming" since they are all based on the principle of stepping forward in time. Many authors (including this author) have assumed that if you cannot do "backward dynamic programming" then you need to turn to "approximate dynamic programming" (which means forward approximate dynamic programming).

While we will see that forward ADP methods can be quite powerful, we are going to first present the idea of backward approximate dynamic programming, which has received comparatively little attention. In fact, we are going to show that backward ADP methods can work much better than forward methods on certain problem classes where classical exact backward dynamic programming does not work.

We will present this idea in several stages:

1) Lookup tables with sampled states - The core idea of backward ADP is to avoid enumerating the entire state space by using a sampled set of states instead. In this first stage, we will still use a lookup table representation of the value functions, and we will also assume we can do full expectations, and maximize over all actions (which generally means a not-too-large set of discrete actions).

2) Sampled expectations - Here we are going to replace the exact expectation with a sampled approximation.

3) Sampled actions - If the number of actions is too large, we can replace the maximization over actions with a maximization over a sampled set.

4) Parametric approximations of the value function - Here we replace the lookup table representation of the value function with a parametric approximation.

16.1 SAMPLED STATES WITH LOOKUP TABLES

The basic idea of backward approximate dynamic programming is to perform classical backward dynamic programming as described in chapter 14, but instead of enumerating all the states $\mathcal{S}$, we work with a sampled set $\hat{\mathcal{S}}$. We illustrate the basic strategy of using a sampled set of states by breaking Bellman's equation into two steps (illustrated in figure 16.2), where the sampled states are shown in gray. Instead of using our basic transition function $S_{t+1} = S^M(S_t, x_t, W_{t+1})$ to describe the transition from pre-decision state S_t to the next pre-decision state S_{t+1}, we use the function $S_t^x = S^{M,x}(S_t, x_t)$ to model the transition from pre-decision state S_t to post-decision state S_t^x, and the function $S_{t+1} = S^{M,x}(S_t^x, W_{t+1})$ to model the transition from post-decision state S_t^x to the next pre-decision state S_{t+1}. Pre- and post-decision states are discussed in more detail in section 9.3.4.

We are also going to make the simplifying assumption (true for some, but hardly all, applications) that the post-decision state space $\mathcal{S}^x$ is "not too large." By contrast, we are going to allow the pre-decision state space $\mathcal{S}$ to be arbitrarily large. This situation arises frequently when there is information needed to make a decision, but which is no longer needed once a decision has been made. Some examples where this arises are:

■ **EXAMPLE 16.1**

As a car traverses from node i to node j on a transportation network, it incurs random costs $\hat{c}_{ij}$ which it learns when it first arrives at node i. The (pre-decision) state when it arrives at node i is then $S = (i, (\hat{c}_{ij})_j)$. After making the decision to traverse from i to some node j' (but before moving to j'), the post-decision state is $S^x = (j)$, since we no longer need the realization of the costs $(\hat{c}_{ij})_j$.

■ **EXAMPLE 16.2**

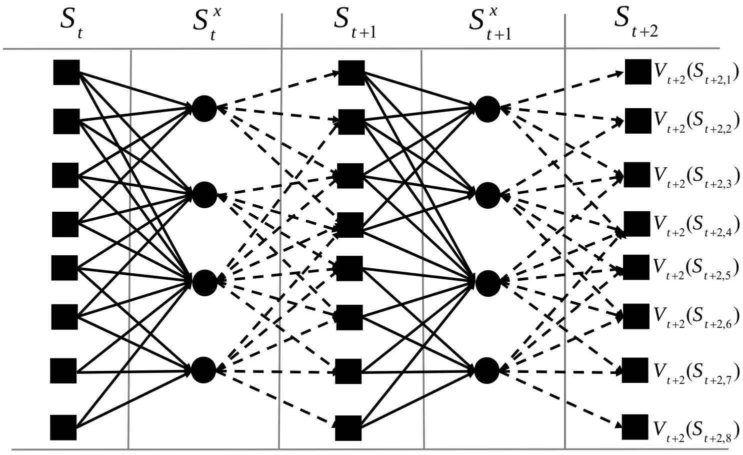


Figure 16.2 Illustration of transitions from pre-decision S_t to post-decision S_t^x to pre-decision S_{t+1} and so on..

A truck driver arrives in city i and learns a set $\mathcal{L}_i$ of loads that need to be moved to other cities. This means when it arrives at i that the state of our driver is $S = (i, \mathcal{L}_i)$. Once the driver chooses a load $\ell \in \mathcal{L}_i$, but before moving to the destination of load ℓ, the (post-decision) state is $S^x = (\ell)$ (or we might use the destination of load ℓ).

■ **EXAMPLE 16.3**

A cement truck is given a set of orders to deliver set to a set of work sites. Let R_t be the inventory of cement, and let $\mathcal{D}_t$ be the set of construction sites needing deliveries (the set includes how much cement is needed by each site). The decision that needs to be made by the cement plant is how much cement to make to replenish inventory. The pre-decision state is $S_t = (R_t, \mathcal{D}_t)$, while the post-decision state is $S_t^x = R_t^x$ which is the amount of inventory left over after making all the deliveries.

In each of these examples, the number of pre-decision states may be extremely large. Instead of looping over all states in $\mathcal{S}$ (as we had to do in figure 16.1), we are going to take a sample $\hat{\mathcal{S}}$ which is of manageable size.

The steps of the algorithm are described in detail in figure 16.4, but we refer to figure 16.3 to explain the idea. The pre-decision states are depicted as squares while post-decision states are circles. We represent the states in our sampled set $\hat{\mathcal{S}}$ using the gray-hatched squares. Assuming we know $\overline{V}_{t+2}(s)$ for states $s \in \hat{\mathcal{S}}$, we compute the value of each post-decision state $\overline{V}^x_{t+1}(s)$ for each post-decision state s in $\mathcal{S}^x$ by taking the expectation over all random outcomes that take us to states in our sampled set $\hat{\mathcal{S}}$. Since not all states are in $\hat{\mathcal{S}}$, when we sum the probabilities over outcomes that take us to states in $\hat{\mathcal{S}}$, these probabilities may not sum to 1.0, so we have to normalize the expected value.

This quickly raises a potential problem. What if none of the random outcomes take us to states in $\hat{\mathcal{S}}$? When this happens, we choose a subset of random outcomes from a

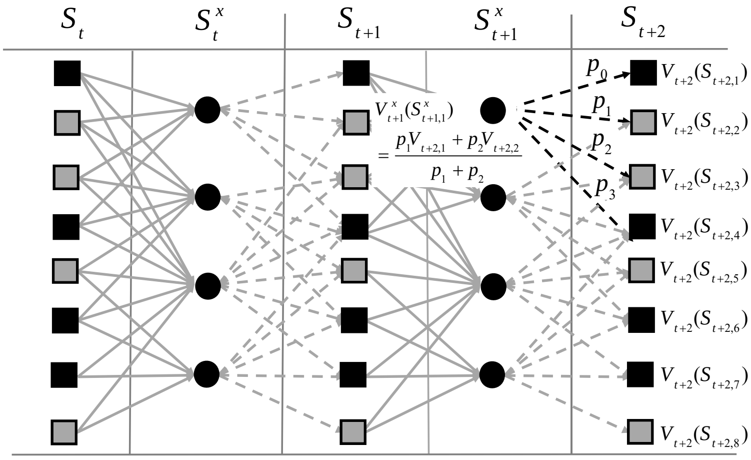


Figure 16.3 Calculation of value of post-decision state S_{t+1}^x using full expectation.

post-decision state, find the pre-decision states that these outcomes take us to, and then add these states to the sampled set $\hat{\mathcal{S}}$. We then repeat the calculation.

Once we have the value of being in each post-decision state, we then step back to find the value of being in each sampled pre-decision state, which is depicted in figure 16.5. Since we assume we have computed the value of being in each post-decision state, finding the value of being in any pre-decision state involves simply searching over all actions and finding the action with the highest one-period reward plus downstream value.

This algorithm can easily encounter two problems, each of which can be solved using a sampling strategy:

1) What if the exogenous information W is continuous and/or multidimensional? We can circumvent this by choosing a sampled set $\hat{\mathcal{W}}_t(s_t^x) = \{w_1, \ldots, w_K\}$ for post-decision state s_t^x. Then compute the probability that we reach an element of $\hat{\mathcal{S}}$ using

$$\rho = \frac{1}{K} \sum_{w \in \hat{\mathcal{W}}_t(s_t^x)} \mathbb{1}_{\{S^{M,W}(s_t^x, w) \in \hat{\mathcal{S}}\}}.$$

If $\rho > 0$, which means we reached at least one state in $\hat{\mathcal{S}}$, then we approximate the expected value of being in post-decision state s^x using

$$\overline{V}_t^x(s_t^x) = \frac{1}{K} \sum_{w \in \hat{\mathcal{W}}_t(s)} \overline{V}_{t+1}(S_t = S^M(s_t^x, x, w)).$$

If $\rho = 0$, then this means that none of the outcomes $w \in \hat{\mathcal{W}}$ produce transitions from our post-decision state s_t^x to a state in $\hat{\mathcal{S}}$. We can circumvent this by adding the states that we do reach to the set $\hat{\mathcal{S}}$ and then repeat the process.

2) What if the decision x is continuous and/or multidimensional? We can use the same process we did above for W, but instead sample a set of K decisions $\hat{\mathcal{X}}_t(s_t) =$

```
Step 0. Initialization:
    0a. Initialize the terminal contribution $V_T(S_T)$.
    0b. Create a sampled set of pre-decision states $\hat{\mathcal{S}}$ (we assume we can use this same sample each time period).
    0c. Create a full set of post-decision states $\mathcal{S}^x$ (presumably a manageable size).
    0d. Set $t = T - 1$.
Step 1a. Step backward in time $t = T, T-1, \ldots, 0$:

Compute the value of each post-decision state:

    Step 2a. Initialize pre-decision value function approximation $\overline{V}_t(s) = -M$.
    Step 2b. Loop over the sampled set of pre-decision states $s \in \hat{\mathcal{S}}$.
    Step 2c. Loop over each action $a \in \mathcal{A}(s)$:
        Step 3a. Compute $Q_t(s,a) = C(s,a) + \overline{V}^x_t(s' = s^{M,x}(s,a))$.
        Step 3b. If $Q_t(s,a) > \overline{V}_t(s)$ then set $\overline{V}_t(s) = Q_t(s,a)$.

Compute the value of each sampled pre-decision state:

    Step 4a. Loop over the full set of post-decision states $s^x \in \mathcal{S}^x$.
    Step 4b. Step back in time: $t = t - 1$.
    Step 4b. Initialize post-decision value function approximation $\overline{V}^x_t(s) = -M$.
        Step 4a. Initialize $Q(s,a) = 0$.
        Step 4b. Initialize total probability $\rho = 0$.
        Step 4c. Loop over each $w \in \mathcal{W}$:
            Step 5a. Compute $Q_t(s,a) = Q_t(s,a) + \mathbb{P}(w|s,a)\overline{V}_{t+1}(s' = s^M(s,a,w))$.
            Step 5b. $\rho = \rho + \mathbb{P}(w|s,a)$.
        Step 4d. If $\rho > 0$ then (we have to normalize $Q_t(s,a)$ in case $\rho < 1$):
            Step 6a. $Q_t(s,a) = Q_t(s,a)/\rho$
        Else: Get here if $\rho = 0$, which means there were no random transitions to states in $\hat{\mathcal{S}}$:
            Step 6b. Choose a sample of outcomes $\hat{w}$ (at least one), find the downstream pre-decision state $\hat{s} = S^{M,W}(s,\hat{w})$, and add each $\hat{s}$ to $\hat{\mathcal{S}}$.
            Step 6c. Return to step 4a.
Step 1b. Return the values $\overline{V}_t(s)$ for all $s \in \mathcal{S}$ and $t = 0, \ldots, T$.
```

Figure 16.4 A backward dynamic programming algorithm using lookup tables.

$\{x_1, \ldots, x_K\}$ from pre-decision state s_t. Then, we solve the maximization problem

$$\hat{v}_t = \max_{x \in \hat{\mathcal{X}}_t(s_t)} \big(C(s_t, x) + \overline{V}^x_t(x^x_t = S^{M,x}(s_t, x))\big).$$

We are not aware of any experimental work with these ideas, so we anticipate that both would benefit from putting some care into how these samples are created. At a minimum, experimentation would be required to determine how large these samples need to be.

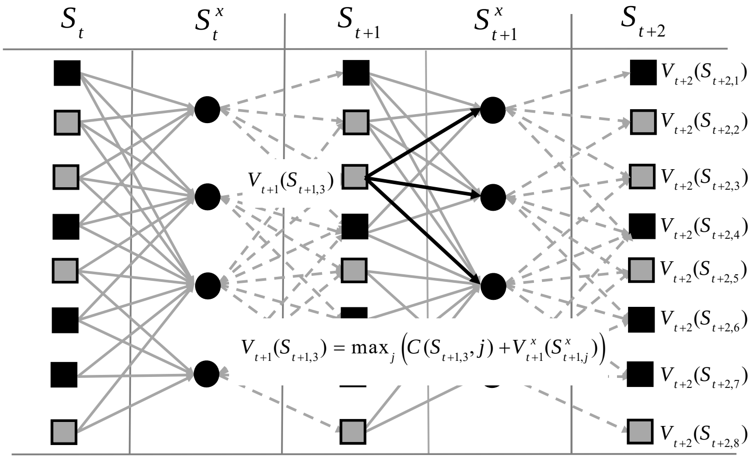


Figure 16.5 Calculation of value of pre-decision state S_{t+1} using full maximization.

16.2 VALUE FUNCTION APPROXIMATION STRATEGIES

We illustrated the basic idea of backward approximate dynamic programming using a standard lookup table representation for the value function, but this would quickly cause problems if we have a multidimensional state (the classic curse of dimensionality). In this section, we suggest three strategies for approximating value functions that mitigate this problem to some degree.

16.2.1 Monotone functions

There are a number of sequential decision problems where the state variable has three to six or seven dimensions, which tends to be the range where the state space is too large to estimate value functions using lookup tables. There are, however, a number of applications where the value function is monotone in each dimension, which is to say that as the state variable increases in each dimension, so does the value of being in the state. Some examples include:

- Optimal replacement of parts and equipment tend to exhibit value functions which are monotone in variables describing the age and/or condition of the parts.

- The problem of controlling the number of patients enrolled in clinical trials produces value functions that are monotone in variables such as the number of enrolled patients, the efficacy of the drug, and the rate at which patients drop out of the study.

- Initiation of drug treatments (statins for cholesteral, metformin for lowering blood sugar) result in value functions that are monotone in health metrics such as cholesterol or blood sugar, the age of a patient and their weight.

- Economic models of expenditures tend to be monotone in the resources available (e.g. personal savings), and other indices such as stock market, interest rates, and unemployment.

Monotonicity can be exploited when we are using a lookup table representation of a value function. Assume that a state s consists of four dimensions $(s_{t1}, s_{t2}, s_{t3}, s_{t4})$, where each dimension takes on one of a set of discrete values, such as $s_{t2} \in \{s_{t21}, s_{t22}, s_{t23}, \ldots, s_{t2J_2}\}$. Assume we have a sampled estimate of the value of being in state s_t^n, which we might compute using

$$\hat{v}_t^n(s_t^n) = \max_x \left(C(s_t^n, x) + \mathbb{E}_{W_t}\{\overline{V}_t^{n-1}(S_{t+1})|s_t^n\}\right),$$

where $S_{t+1} = S^M(s_t^n, x, W_{t+1})$. We might then use our sampled estimate (regardless of how it is found) to update the value function approximation at state s_t^n using

$$\overline{V}_t^n(s_t^n) = (1-\alpha_n)\overline{V}_t^{n-1}(s_t^n) + \alpha_n \hat{v}_t^n(s_t^n).$$

We assume that $\overline{V}_t^{n-1}(s)$ is monotone in s before the update. Assume that $s' \succ s$ means that each element $s'_{ij} \geq s_{ij}$. Then if $\overline{V}_t^{n-1}(s)$ is monotone in s, then $s' \succ s$ means that $\overline{V}_t^{n-1}(s') \geq \overline{V}_t^{n-1}(s)$. However, we cannot assume that this is true of $\overline{V}_t^n(s)$ just after we have done an update for state s_t^n. We can quickly check if $\overline{V}_t^n(s) \leq \overline{V}_t^n(s')$ for each s' with at least one element that is larger than the corresponding element of s.

The idea is illustrated in 16.6. Starting with the upper left corner, we start with an initial value function $\overline{V}(s) = 0$, and make an observation (the blue dot) of 10 in the middle. We then use the monotone structure to make all points to the right and above of this point to equal 10. We then make an observation of 5, and use this observation to update all the points to the left and below the last observation.

Figure 16.7 shows snapshots from a video where monotonicity is being used to update a two-dimensional function. Again starting from the upper right, the first three screenshots were from the first 20 iterations, while the last one (lower right) was at the end, long after the function had stopped changing. The use of monotonicity

Monotonicity is an important structural property. When it holds, it dramatically speeds the process of learning the value functions. We have used this idea for matrices with as many as seven dimensions, although at that point a lookup representation of a seven-dimensional function becomes quite large.

There will be situations where a value function is monotone in some dimensions, but not in others. This can be handled (somewhat clumsily) but imposing monotonicity over the subset of states where monotonicity holds. For the remaining states, we have to resort to brute force lookup table methods. If $\bar{s}$ is the set of states where the value function is not monotone, while $\tilde{s}$ is the states over which the value function is monotone (of course, $s = (\tilde{s}, \bar{s})$), then we can think of a value function $\overline{V}(\tilde{s}, \bar{s})$ where we have a monotone value function $\overline{V}(\tilde{s}|\bar{s})$ for each state $\bar{s}$ (we hope there are not too many of these).

16.2.2 Linear models

Arguably the most natural strategy for approximating the value function is to fit a statistical model, where the most natural starting point is a linear model of the form

$$\overline{V}_t(S_t|\theta_t) = \sum_{f \in \mathcal{F}} \theta_{tf}\phi_f(S_t).$$

Here, $\phi_f(S_t)$ are a set of appropriately chosen features. For example, if S_t is a continuous scalar (such as price), we might use $\phi_1(S_t) = S_t$ and $\phi_2(S_t) = S_t^2$.

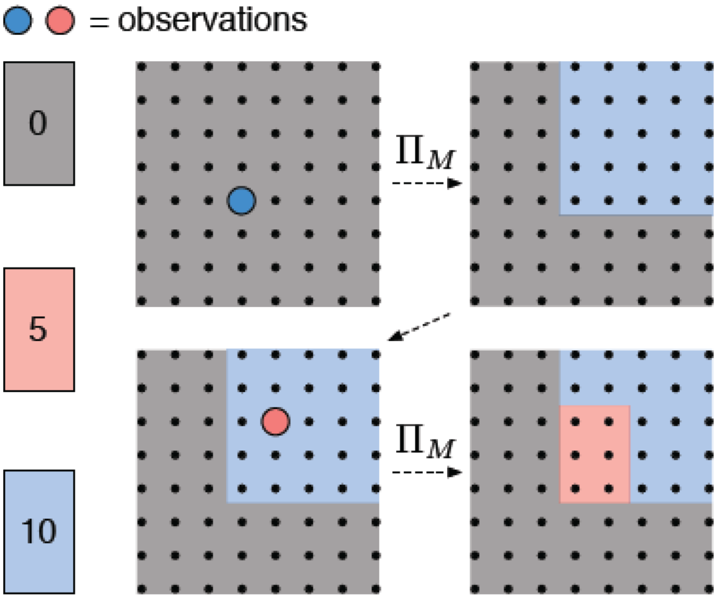


Figure 16.6 Illustration of the use of monotonicity. Starting from upper left: 1) Initial value function all 0, with observation (blue dot) of 10; 2) using observation to update all points to the right and above to 10; 3) new observation (pink dot) of 5; 4) updating all points to the left and below to 5. (Graphic due to Daniel Jiang)

The idea is very simple. For each $\hat{s}$ in our sampled set of pre-decision states $\hat{\mathcal{S}}$, compute a sampled estimate $\hat{v}_t^n$ of the value of being in a state s_t^n

$$\hat{v}_t^n = \arg\max_x \left(C(\hat{s}_t^n, x) + \mathbb{E}\{\overline{V}_{t+1}(S_{t+1})|S_t\} \right),$$

where $S_{t+1} = S^M(\hat{s}_t^n, x, W_{t+1})$.

Now, we have a set of data $(\hat{v}_t^n, \hat{s}_t^n)$ for $n = 1, \ldots, |\hat{\mathcal{S}}|$. We can use this dataset to estimate any statistical model $\overline{V}_t(S_t|\theta_t)$ which gives us an estimate of the value of being in every state, not just the sampled states. For example, assume we have a linear model (remember this means linear in the parameters)

$$\begin{aligned}\overline{V}_t(S_t|\bar{\theta}_t) &= \bar{\theta}_{t1}\phi_1(S_t) + \bar{\theta}_{t2}\phi_2(S_t) + \bar{\theta}_{t3}\phi_3(S_t) + \ldots, \\ &= \sum_{f \in \mathcal{F}} \theta_{tf}\phi_f(S_t),\end{aligned}$$

where $\phi_f(S_t)$ is some feature of the state. This might be the inventory R_t (money in the bank, units of blood), or R_t^2, or $\ln(R_t)$. Create the (column) vector ϕ^n using

$$\phi^n = \begin{pmatrix} \phi_1^n \\ \phi_2^n \\ \vdots \\ \phi_F^n \end{pmatrix}$$

where $\phi_f^n = \phi_f(S_t^n)$.

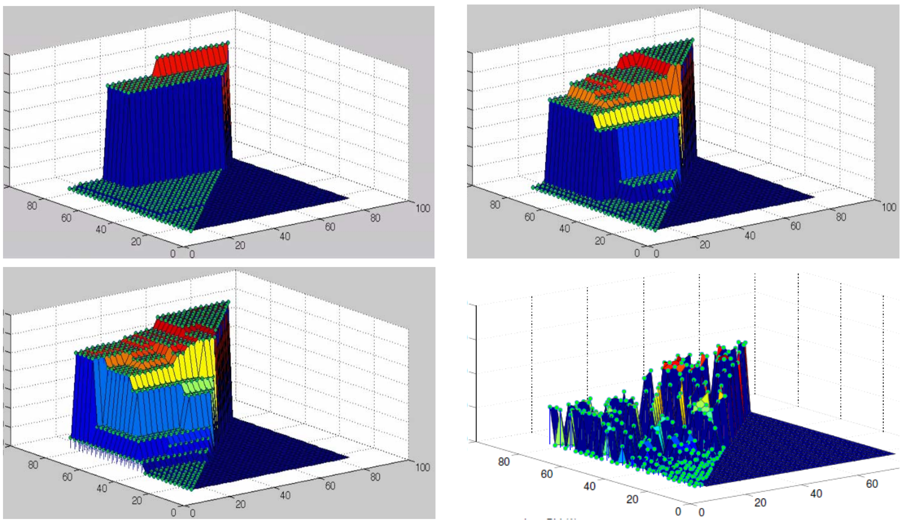


Figure 16.7 Video snapshot of use of monotonicity for a two-dimensional function for three updates; fourth snapshot (lower right) is a value function where monotonicity was not used.

Let $\hat{v}_t^n$ be computed using (16.2), which we can think of as a sample realization of the estimate $\overline{V}_t^{n-1}(S_t)$. We can think of

$$\hat{\varepsilon}_t^n = \overline{V}_t^{n-1}(S_t) - \hat{v}_t^n$$

as the "error" in our estimate. Using the methods we first introduced in section 3.8.1, we can update our estimates of the parameter vector $\bar{\theta}_t^{n-1}$ using

$$\bar{\theta}_t^n = \bar{\theta}_t^{n-1} - H_t^n \phi_t^n \hat{\varepsilon}_t^n, \tag{16.2}$$

where H_t^n is a matrix computed using

$$H_t^n = \frac{1}{\gamma^n} B_t^{n-1}. \tag{16.3}$$

B_t^{n-1} is an $|\mathcal{F}|$ by $|\mathcal{F}|$ matrix which is updated recursively using

$$B_t^n = B_t^{n-1} - \frac{1}{\gamma_t^n}(B_t^{n-1}\phi_t^n(\phi_t^n)^T B_t^{n-1}). \tag{16.4}$$

γ_t^n is a scalar computed using

$$\gamma_t^n \quad = \quad 1 + (\phi_t^n)^T B_t^{n-1} \phi_t^n. \tag{16.5}$$

Parametric approximations are particularly attractive because we get an estimate of the value of being in *every* state from a small sample. The price we pay for this generality is the errors introduced by our parametric approximation.

16.2.3 Other approximation models

We encourage readers to experiment with other methods from chapter 3 (or your favorite book in statistics or machine learning). We note that approximation errors will accumulate with backward ADP, so you should not have much confidence that $\overline{V}_t(S_t)$ is actually a good approximation of the value of being in state S_t. However, we have found that even when there is a significant difference between $\overline{V}_t(S_t)$ and the true value function $V_t(S_t)$ (when we can find this), the approximation $\overline{V}_t(S_t)$ may still provide a high quality policy (but not always).

16.3 NUMERICAL APPROXIMATION METHODS

There are many problems that are hard primarily because the state, information and controls are continuous. These problems may be multidimensional, but typically are relatively low dimensional (one to five dimensions is fairly typical).

16.4 COMPUTATIONAL NOTES

Some thoughts to keep in mind while designing and testing algorithms using backward approximate dynamic programming:

Approximation architectures It is possible to use any of the statistical learning methods described in chapter 3.8.1 (or your favorite book on statistics/machine learning). We note that most of the methods in this book involve adaptive learning (this is the focus of chapter 3.8.1), but with backward ADP, we actually return to the more familiar setting (in the statistical learning community) of batch learning. Following standard advice in the specification of any statistical model, make sure that the dimensionality of the model (measured by the number of parameters) is much smaller than the number of datapoints to avoid overfitting.

Tuning Virtually all adaptive learning algorithms have tunable parameters, and this is the Achilles heel of this entire approach to solving stochastic optimization problem. In chapter 9, section 9.10 summarizes four problem classes (see table 9.2), we describe four problem classes, where classes (1) and (4) are posed as finding the best learning policy. These "learning policies" represent the process of finding the best search algorithm, which includes tuning the parameters that govern a particular class of algorithm. In practice, this search for the best learning policy (or equivalently, the search for the best search algorithm) is typically done in an ad hoc way. There are thousands of papers which will prove asymptotic convergence, but the actual design of an algorithm depends on ad hoc testing.

Validating A major challenge with any approximation strategy, backward ADP included, is validation. Backward ADP can work extremely well on problems where the value function is a fairly good approximation of the true value function, but there are no guarantees.

Performance We have obtained exceptionally good performance on some problem classes, including energy storage problems with thousands of time periods. In comparisons against optimal policies (obtained using the methods from chapter 14 for low-dimensional problem instances), we have obtained solutions that were over 95 percent of optimality, but on occasions the performance was as low as 70 percent when we did a poor job with the approximations.

16.5 BIBLIOGRAPHIC NOTES

- Section xx -

CHAPTER 17

FORWARD ADP I: THE VALUE OF A POLICY

In chapter 11, we described a number of different ways of constructing a policy. One of the most important ways, and the way that is most widely associated with the term “approximate dynamic programming,” requires approximating the value of being in a state. Chapter 3 described a number of different approximation strategies, including lookup table, aggregated functions, parametric models and nonparametric models. All of these statistical models are created by generating a state S^n, next computing some observation of a value $\hat{v}^n$ of being in state S^n, and finally using the pair $(S^n, \hat{v}^n)$ to estimate (or update the estimate) of the value of being in a state.

In this chapter, we focus primarily on the different ways of calculating $\hat{v}^n$, and then using this information to estimate a value function approximation, *for a fixed policy*. To emphasize that we are computing values for a fixed policy, we index parameters such as the value function V^π by the policy π. After we establish the fundamentals for estimating the value of a policy, chapter 18 addresses the last step of searching for good policies.

17.1 SAMPLING THE VALUE OF A POLICY

On first glance, the problem of statistically estimating the value of a fixed policy should not be any different than estimating a function from noisy observations. We start by showing that from one perspective, this is precisely correct. However, the context of dynamic programming, even when we fix a policy, introduces opportunities and challenges when we realize that we can take advantage of the dynamics of information, which may arise in both finite and infinite horizon settings.

Stochastic Optimization and Learning. By Warren B. Powell
Copyright © 2019 John Wiley & Sons, Inc.

```
Step 0. Initialization:
    Step 0a. Initialize $\overline{V}^0$.
    Step 0b. Initialize $S^1$.
    Step 0c. Set $n = 1$.
Step 1. Choose a sample path $\omega^n$.
Step 2. Choose a starting state $S_0^n$.
Step 3. Do for $t = 0, 1, \ldots, T$:
    Step 3a. $a_t^n = A^\pi(S_t^n)$.
    Step 3b. $\hat{C}_t^n = C(S_t^n, a_t^n)$.
    Step 3c. $W_{t+1}^n = W_{t+1}(\omega^n)$.
    Step 3d. $S_{t+1}^n = S^M(S_t^n, a_t^n, W_{t+1}^n)$.
Step 4. Compute $\hat{v}_0^n = \sum_{t=0}^T \gamma^t \hat{C}_t^n$.
Step 5. Increment $n$. If $n \leq N$ go to Step 1.
Step 6. Use the sequence of state-value pairs $(S^i, \hat{v}^i)_{i=1}^N$ to fit a value function approximation $\overline{V}^\pi(s)$.
```

Figure 17.1 Basic policy approximation method.

17.1.1 Direct policy evaluation for finite horizon problems

Imagine that we have a fixed policy $A^\pi(s)$ (or $X^\pi(s)$ if we are working with vector-valued decisions). The policy may take any of the forms described in chapter 11. For iteration n, if we are in state S_t^n at time t, we then choose action $a_t^n = A^\pi(S_t^n)$, after which we sample the exogenous information W_{t+1}^n. We sometimes say that we are following sample path ω^n from which we observe $W_{t+1}^n = W_{t+1}(\omega^n)$. The exogenous information W_{t+1}^n may depend on both S_t^n and the action a_t^n. From this, we may compute our contribution (cost if we are minimizing) from

$$\hat{C}_t^n = C(S_t^n, a_t^n, W_{t+1}^n).$$

Finally, we compute our next state from our transition function

$$S_{t+1}^n = S^M(S_t^n, a_t^n, W_{t+1}^n).$$

This process continues until we reach the end of our horizon T. The basic algorithm is described in figure 17.1. In step 6, we use a batch routine to fit a statistical model. It is often more natural to use some sort of recursive procedure and imbed the updating of the value function within the iterative loop. The type of recursive procedure depends on the nature of the value function approximation. Later in this chapter, we describe several recursive procedures if we are using linear regression.

Finite horizon problems are sometimes referred to as *episodic*, where an episode refers to a simulation of a policy until the end of the horizon (also known as trials). However, the term episodic can also be interpreted more broadly. For example, an emergency vehicle may repeatedly return to base where the system then restarts. Each cycle of starting from a home base and then returning to the home base can be viewed as an episode. As a result, if we are working with a finite horizon problem, we prefer to refer to these specifically as such.

Evaluating a fixed policy is mathematically equivalent to making unbiased observations of a noisy function. Fitting a functional approximation is precisely what the entire field of statistical learning has been trying to do for decades. If we are fitting a linear model, then there are some powerful recursive procedures that can be used. These are discussed below.

17.1.2 Policy evaluation for infinite horizon problems

Not surprisingly, infinite horizon problems introduce a special complication, since we cannot obtain an unbiased observation in a finite number of measurements. Below we present some methods that have been used for infinite horizon applications.

Recurrent visits

There are many problems which are infinite horizon, but where the system resets itself periodically. A simple example of this is a finite horizon problem, where hitting the end of the horizon and starting over (as would occur in a game) can be viewed as an episode. A different example is a queueing system, where perhaps we are trying to manage the admission of patients to an emergency room. From time to time the queue may become empty, at which point the system resets and starts over. For such systems, it makes sense to estimate the value of following a policy π when starting from this base state.

Even if we do not have such a renewal system, imagine that we find ourselves in a state s. Now follow a policy π until we re-enter state s again. Let $R^n(s)$ be the reward earned, and let $\tau^n(s)$ be the number of time periods required before re-entering state s. Here, n is counting the number of times we visit state s. An observation of the average reward earned when in state s and following policy π would be given by

$$\hat{v}^n(s) = \frac{R^n(s)}{\tau^n(s)}.$$

$\hat{v}^n(s)$ would be computed when we return to state s. We might then update the *average* value of being in state s using

$$\bar{v}^n(s) = (1 - \alpha_{n-1})\bar{v}^{n-1}(s) + \alpha_{n-1}\hat{v}^n(s).$$

Note that as we make each transition from some state s' to some state s'', we are accumulating rewards in $R(s)$ for every state s that we have visited prior to reaching state s'. Each time we arrive at some state s'', we stop accumulating rewards for s'', and compute $\hat{v}^n(s'')$, and then smooth this into the current estimate of $\bar{v}(s'')$. Note that we have presented this only for the case of computing the average reward per time period.

Partial simulations

While we may not be able to simulate an infinite trajectory, we may simulate a long trajectory T, long enough to ensure that we are producing an estimate that is "good enough." When we are using discounting, we realize that eventually γ^t becomes small enough that a longer simulation does not really matter. This idea can be implemented in a relatively simple way.

Consider the algorithm in figure 17.1, and insert the calculation in step 3:

$$\bar{c}_t = \frac{t-1}{t}\bar{c}_{t-1} + \frac{1}{t}\hat{C}_t^n.$$

$\bar{c}_t$ is an average over the time periods of the contribution per time period. As we follow our policy over progressively more time periods, $\bar{c}_t$ approaches an average contribution per time period. Over an infinite horizon, we would expect to find

$$\hat{v}_0^n = \lim_{t \to \infty} \sum_{t=0}^{\infty} \gamma^t \hat{C}_t^n = \frac{1}{1-\gamma} \bar{c}_\infty.$$

Now assume that we only progress T time periods, and let $\bar{c}_T$ be our estimate of $\bar{c}_\infty$ at this point. We would expect that

$$\begin{aligned} \hat{v}_0^n(T) &= \sum_{t=0}^{T} \gamma^t \hat{C}_t^n \\ &\approx \frac{1-\gamma^{T+1}}{1-\gamma} \bar{c}_T. \end{aligned} \tag{17.1}$$

The error between our T-period estimate $\hat{v}_0^n(T)$ and the infinite horizon estimate $\hat{v}_0^n$ is given by

$$\begin{aligned} \delta_T^n &= \frac{1}{1-\gamma} \bar{c}_\infty - \frac{1-\gamma^{T+1}}{1-\gamma} \bar{c}_T \\ &\approx \frac{1}{1-\gamma} \bar{c}_T - \frac{1-\gamma^{T+1}}{1-\gamma} \bar{c}_T \\ &= \frac{\gamma^{T+1}}{1-\gamma} \bar{c}_T. \end{aligned}$$

Thus, we just have to find T to make δ_T small enough. This strategy is imbedded in some optimal algorithms, which only require that $\delta_T^n \to 0$ as $n \to \infty$ (meaning that we have to steadily allow T to grow).

Infinite horizon projection

The analysis above leads to another idea that has received considerable attention in the approximate dynamic programming community under the name *least squares temporal differencing* (LSTD), although we present it here with a somewhat different development. We can easily see from (17.1) that if we stop after T time periods, we will underestimate the infinite horizon contribution by a factor $1 - \gamma^{T+1}$. Assuming that T is reasonably large (say, $\gamma^{T+1} < 0.1$), we might introduce the correction

$$\hat{v}_0^n = \frac{1}{1-\gamma^{T+1}} \hat{v}_0^n(T).$$

In essence we are taking a sample estimate of a T-period path, and projecting it out over an infinite horizon.

17.1.3 Temporal difference updates

Assume that we are in state S_t^n and we make decision a_t^n (using policy π), after which we observe the information W_{t+1} which puts us in state $S_{t+1}^n = S^M(S_t^n, a_t^n, W_{t+1}^n)$. The contribution from this transition is given by $C(S_t^n, a_t^n)$ (or $C(S_t^n, a_t^n, W_{t+1}^n)$ if the contribution depends on the outcome W_{t+1}). Imagine now that we continue this until the end of our horizon T. For simplicity, we are going to drop discounting. In this case, the contribution along this path would be

$$\hat{v}_t^n = C(S_t^n, a_t^n) + C(S_{t+1}^n, a_{t+1}^n) + \ldots + C(S_T^n, a_T^n). \tag{17.2}$$

This is the contribution from following the path produced by a combination of the information from outcome ω^n (this determines $W_{t+1}^n, W_{t+2}^n, \ldots, W_T^n$) and policy π. $\hat{v}_t^n$ is an unbiased sample estimate of the value of being in state S_t and following policy π over sample path ω^n. We can use a stochastic gradient algorithm to estimate the value of being in state S_t using

$$\overline{V}_t^n(S_t^n) \quad = \quad \overline{V}_t^{n-1}(S_t^n) - \alpha_n \left(\overline{V}_t^{n-1}(S_t^n) - \hat{v}_t^n \right). \tag{17.3}$$

We can obtain a richer class of algorithms by breaking down our path cost in (17.2) by using

$$\begin{aligned}\hat{v}_t^n \quad = \quad & \sum_{\tau=t}^{T} C(S_\tau^n, a_\tau^n, W_{\tau+1}^n) \\ & \underbrace{- \left\{ \sum_{\tau=t}^{T} \left(\overline{V}_\tau^{n-1}(S_\tau) - \overline{V}_{\tau+1}^{n-1}(S_{\tau+1}) \right) \right\} + (\overline{V}_t^{n-1}(S_t) - \overline{V}_{T+1}^{n-1}(S_{T+1})).}_{=0}\end{aligned}$$

We now use the fact that $\overline{V}_{T+1}^{n-1}(S_{T+1}) = 0$ (this is where our finite horizon model is useful). Rearranging gives

$$\hat{v}_t^n \quad = \quad \overline{V}_t^{n-1}(S_t) + \sum_{\tau=t}^{T} \left(C(S_\tau^n, a_\tau^n, W_{\tau+1}^n) + \overline{V}_{\tau+1}^{n-1}(S_{\tau+1}) - \overline{V}_\tau^{n-1}(S_\tau) \right).$$

Let

$$\delta_\tau \quad = \quad C(S_\tau^n, a_\tau^n, W_{\tau+1}^n) + \overline{V}_{\tau+1}^{n-1}(S_{\tau+1}^n) - \overline{V}_\tau^{n-1}(S_\tau^n). \tag{17.4}$$

The terms δ_τ are called *temporal differences*. If we were using a standard single-pass algorithm, then at time t, $\hat{v}_t^n = C(S_t^n, a_t^n, W_{t+1}^n) + \overline{V}_{t+1}^{n-1}(S_{t+1}^n)$ would be our sample observation of being in state S_t, while $\overline{V}_t^{n-1}(S_t)$ is our current estimate of the value of being in state S_t. This means that the temporal difference at time t, $\delta_t = \hat{v}_t^n - \overline{V}_t^{n-1}(S_t)$, is the difference in our estimate of the value of being in state S_t between our current estimate and the updated estimate. The temporal difference is also known as the *Bellman error*.

Using (17.4), we can write $\hat{v}_t^n$ in the more compact form

$$\hat{v}_t^n = \overline{V}_t^{n-1}(S_t) + \sum_{\tau=t}^{T} \delta_\tau. \tag{17.5}$$

Substituting (17.5) into (17.3) gives

$$\begin{aligned}\overline{V}_t^n(S_t) \quad & = \quad \overline{V}_t^{n-1}(S_t) - \alpha_{n-1} \left[\overline{V}_t^{n-1}(S_t) - \left(\overline{V}_t^{n-1}(S_t) + \sum_{\tau=t}^{T} \delta_\tau \right) \right] \\ & = \quad \overline{V}_t^{n-1}(S_t) + \alpha_{n-1} \sum_{\tau=t}^{T-1} \delta_\tau. \qquad (17.6)\end{aligned}$$

We next use this bit of algebra to build an important class of updating mechanisms for estimating value functions.

17.1.4 TD(λ)

The temporal differences δ_τ are the errors in our estimates of the value of being in state S_τ. We can think of each term in (17.6) as a correction to the estimate of the value function. It makes sense that updates farther along the path should not be given as much weight as those earlier in the path. As a result, it is common to introduce an artificial discount factor λ, producing updates of the form

$$\overline{V}_t^n(S_t) \quad = \quad \overline{V}_t^{n-1}(S_t) + \alpha_{n-1} \sum_{\tau=t}^{T} \lambda^{\tau-t} \delta_\tau. \tag{17.7}$$

We derived this formula without a time discount factor. We leave as an exercise to the reader to show that if we have a time discount factor γ, then the temporal-difference update becomes

$$\overline{V}_t^n(S_t) \quad = \quad \overline{V}_t^{n-1}(S_t) + \alpha_{n-1} \sum_{\tau=t}^{T} (\gamma\lambda)^{\tau-t} \delta_\tau. \tag{17.8}$$

Equation (17.8) shows that the discount factor γ, which is typically viewed as capturing the time value of money, and the algorithmic discount λ, which is a purely algorithmic device, have exactly the same effect. Not surprisingly, modelers in operations research have often used a discount factor γ set to a much smaller number than would be required to capture the time-value of money. Artificial discounting allows us to look into the future, but then discount the results when we feel that the results are not perfectly accurate. Note that our use of λ in this setting is precisely equivalent to our discounting (also using λ) when we presented discounted rolling horizon policies in section 20.4.3.

Updates of the form given in equation (17.7) produce an updating procedure that is known as TD(λ) (or, temporal difference learning with discount λ). We have seen this form of discounting in section 20.3.4, when we first introduced λ as a form of algorithmic discounting.

The updating formula in equation (17.7) requires that we step all the way to the end of the horizon before updating our estimates of the value. There is, however, another way of implementing the updates. The temporal differences δ_τ are computed as the algorithm steps forward in time. As a result, our updating formula can be implemented recursively. Assume we are at time t' in our simulation. We would simply execute

$$\overline{V}_t^n(S_t^n) := \overline{V}_t^n(S_t) + \alpha_{n-1}\lambda^{t'-t}\delta_{t'} \text{ for all } t \le t'. \tag{17.9}$$

Here, our notation ":=" means that we take the current value of $\overline{V}_t^n(S_t)$, add $\alpha_{n-1}\lambda^{t'-t}\delta_{t'}$ to it to obtain an updated value of $\overline{V}_t^n(S_t)$. When we reach time $t' = T$, our value functions would have undergone a complete update. We note that at time t', we need to update the value function for every $t \le t'$.

17.1.5 TD(0) and approximate value iteration

An important special case of TD(λ) occurs when we use $\lambda = 0$. In this case,

$$\begin{aligned}\overline{V}^n_t(S^n_t) &= \overline{V}^{n-1}_t(S^n_t) + \alpha_{n-1}\big(C(S^n_t, a^n_t) + \gamma\overline{V}^{n-1}(S^M(S^n_t, a^n_t, W^n_{t+1})) \\ &\quad - \overline{V}^{n-1}_t(S^n_t)\big). \qquad (17.10)\end{aligned}$$

Now consider value iteration. In chapter 14, when we did not have to deal with Monte Carlo samples and statistical noise, value iteration (for a fixed policy) looked like

$$V^n_t(s) = C(s, A^\pi(s)) + \gamma\sum_{s'\in\mathcal{S}} p^\pi(s'|s)V^n_{t+1}(s').$$

In steady state, we would write it as

$$V^n(s) = C(s, A^\pi(s)) + \gamma\sum_{s'\in\mathcal{S}} p^\pi(s'|s)V^{n-1}(s').$$

When we use approximate dynamic programming, we are following a sample path that puts us in state S^n_t, where we observe a sample realization of a contribution $\hat{C}^n_t$, after which we observe a sample realization of the next downstream state S^n_{t+1} (the action is determined by our fixed policy). A sample observation of the value of being in state S^n_t would be computed using

$$\hat{v}^n_t = C(S^n_t, a^n_t) + \gamma\overline{V}^{n-1}_{t+1}(S^n_{t+1}).$$

We can then use this to update our estimate of the value of being in state S^n_t using

$$\begin{aligned}\overline{V}^n_t(S^n_t) &= (1-\alpha_{n-1})\overline{V}^{n-1}_t(S^n_t) + \alpha_{n-1}\hat{v}^n_t \\ &= (1-\alpha_{n-1})\overline{V}^{n-1}_t(S^n_t) \\ &\quad + \alpha_{n-1}\big(C(S^n_t, a^n_t) + \gamma\overline{V}^{n-1}(S^M(S^n_t, a^n_t, W^n_{t+1}))\big). \qquad (17.11)\end{aligned}$$

It is not hard to see that (17.10) and (17.11) are the same. The idea is popular because it is particularly easy to implement. It is also well suited to high-dimensional decision vectors x, as we illustrate in chapter 19.

Temporal difference learning derives its name because $\overline{V}^{n-1}(S)$ is viewed as the "current" value of being in state S, while $C(S,a) + \overline{V}^{n-1}(S^M(S,a,W))$ is viewed as the updated value of being in state S. The difference $\overline{V}^{n-1}(S) - (C(S,a) + \overline{V}^{n-1}(S^M(S,a,W)))$ is the difference in these estimates across iterations (or time), hence the name. TD(0) is a form of statistical bootstrapping, because rather than simulate the full trajectory, it depends on the current estimate of the value $\overline{V}^{n-1}(S^M(S,a,W))$ of being in the downstream state $S^M(S,a,W)$.

While TD(0) can be very easy to implement, it can also produce very slow convergence. The effect is illustrated in figure 17.1, where there are five steps before earning a reward of 1 (which we always earn). In this illustration, there are no decisions and the contribution is zero for every other time period. A stepsize of $1/n$ was used throughout.

The table illustrates that the rate of convergence for $\overline{V}_0$ is dramatically slower than for $\overline{V}_4$. The reason is that as we smooth $\hat{v}_t$ into $\overline{V}_{t-1}$, the stepsize has a discounting effect. The problem is most pronounced when the value of being in a state at time t depends on

Iteration	$\overline{V}_0$	$\hat{v}_1$	$\overline{V}_1$	$\hat{v}_2$	$\overline{V}_2$	$\hat{v}_3$	$\overline{V}_3$	$\hat{v}_4$	$\overline{V}_4$	$\hat{v}_5$
0	0.000		0.000		0.000		0.000		0.000	1
1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	1
2	0.000	0.000	0.000	0.000	0.000	0.000	0.500	1.000	1.000	1
3	0.000	0.000	0.000	0.000	0.167	0.500	0.667	1.000	1.000	1
4	0.000	0.000	0.042	0.167	0.292	0.667	0.750	1.000	1.000	1
5	0.008	0.042	0.092	0.292	0.383	0.750	0.800	1.000	1.000	1
6	0.022	0.092	0.140	0.383	0.453	0.800	0.833	1.000	1.000	1
7	0.039	0.140	0.185	0.453	0.507	0.833	0.857	1.000	1.000	1
8	0.057	0.185	0.225	0.507	0.551	0.857	0.875	1.000	1.000	1
9	0.076	0.225	0.261	0.551	0.587	0.875	0.889	1.000	1.000	1
10	0.095	0.261	0.294	0.587	0.617	0.889	0.900	1.000	1.000	1

Table 17.1 Effect of stepsize on backward learning

contributions that are a number of steps into the future (imagine the challenge of training a value function to play the game of chess). For problems with long horizons, and in particular those where it takes many steps before receiving a reward, this bias can be so serious that it can appear that temporal differencing (and algorithms that use it) simply does not work. We can partially overcome the slow convergence by carefully choosing a stepsize rule. Stepsizes are discussed in depth in chapter 6.

17.1.6 TD learning for infinite horizon problems

We can perform updates using a general TD(λ) strategy as we did for finite horizon problems. However, there are some subtle differences. With finite horizon problems, it is common to assume that we are estimating a different function $\overline{V}_t$ for each time period t. As we step through time, we obtain information that can be used for a value function at a *specific* point in time. With stationary problems, each transition produces information that can be used to update the value function, which is then used in all future updates. By contrast, if we update $\overline{V}_t$ for a finite horizon problem, then this update is not used until the next forward pass through the states.

When we move to infinite horizon problems, we drop the indexing by t. Instead of stepping forward in time, we step through iterations, where at each iteration we generate a temporal difference

$$\delta^n \quad = \quad C(s^n, a^n) + \gamma \overline{V}^{n-1}(S^{M,a}(s^n, a^n)) - \overline{V}^{n-1}(s^n).$$

To do a proper update of the value function at each state, we would have to use an infinite series of the form

$$\overline{V}^n(s) \quad = \quad \overline{V}^{n-1}(s) + \alpha_n \sum_{m=0}^{\infty} (\gamma\lambda)^m \delta^{n+m}, \qquad (17.12)$$

where we can use any initial starting state $s^0 = s$. Of course, we would use the same update for each state s^m that we visit, so we might write

$$\overline{V}^n(s^m) \quad = \quad \overline{V}^{n-1}(s^m) + \alpha_n \sum_{n=m}^{\infty} (\gamma\lambda)^{(n-m)} \delta^n. \qquad (17.13)$$

```
Step 0. Initialization:
    Step 0a. Initialize $\overline{V}^0(S)$ for all $S$.
    Step 0b. Initialize the state $S^0$.
    Step 0c. Set $n = 1$.
Step 1. Choose $\omega^n$.
Step 2. Solve
    $a^n = \arg\max_{a \in \mathcal{A}^n} \big(C(S^n, a) + \gamma \overline{V}^{n-1}(S^{M,a}(S^n, a))\big).$   (17.15)
Step 3. Compute the temporal difference for this step:
    $\delta^n = C(S^n, a^n) + \gamma \left(\overline{V}^{n-1}(S^{M,a}(S^n, a^n)) - \overline{V}^{n-1}(S^n)\right).$
Step 4. Update $\overline{V}$ for $m = n, n-1, \ldots, 1$:
    $\overline{V}^n(S^m) = \overline{V}^{n-1}(S^m) + (\gamma\lambda)^{n-m}\delta^n.$   (17.16)
Step 5. Compute $S^{n+1} = S^M(S^n, a^n, W(\omega^n))$.
Step 6. Let $n = n + 1$. If $n < N$, go to step 1.
```

Figure 17.2 A TD(λ) algorithm for infinite horizon problems.

Equations (17.12) and (17.13) both imply stepping forward in time (presumably a "large" number of iterations) and computing temporal differences before performing an update. A more natural way to run the algorithm is to do the updates incrementally. After we compute δ^n, we can update the value function at each of the previous states we visited. So, at iteration n, we would execute

$$\overline{V}^n(s^m) \quad := \quad \overline{V}^n(s^m) + \alpha_n(\gamma\lambda)^{n-m}\delta^m, \quad m = n, n-1, \ldots, 1. \tag{17.14}$$

We can now use the temporal difference δ^n to update the estimate of the value function for every state we have visited up to iteration n.

Figure 17.2 outlines the basic structure of a TD(λ) algorithm for an infinite horizon problem. Step 1 begins by computing the first post-decision state, after which step 2 makes a single step forward. After computing the temporal-difference in step 3, we traverse previous states we have visited in Step 4 to update their value functions.

In step 3, we update all the states $(S^m)_{m=1}^n$ that we have visited up to then. Thus, at iteration n, we would have simulated the partial update

$$\overline{V}^n(S^0) \quad = \quad \overline{V}^{n-1}(S^0) + \alpha_{n-1}\sum_{m=0}^{n}(\gamma\lambda)^m\delta^m. \tag{17.17}$$

This means that at any iteration n, we have updated our values using biased sample observations (as is generally the case in value iteration). We avoided this problem for finite horizon problems by extending out to the end of the horizon. We can obtain unbiased updates for infinite horizon problems by assuming that all policies eventually put the system into an "absorbing state." For example, if we are modeling the process of holding or selling an asset, we might be able to guarantee that we eventually sell the asset.

One subtle difference between temporal difference learning for finite horizon and infinite horizon problems is that in the infinite horizon case, we may be visiting the same state two or more times on the same sample path. For the finite horizon case, the states and value functions are all indexed by the time that we visit them. Since we step forward through time, we can never visit the same state at the same point in time twice in the same sample path. By contrast, it is quite easy in a steady-state problem to revisit the same state over and over again. For example, we could trace the path of our nomadic trucker, who might go back and forth between the same pair of locations in the same sample path. As a result, we are using the value function to determine what state to visit, but at the same time we are updating the value of being in these states.

17.2 STOCHASTIC APPROXIMATION METHODS

A central idea in recursive estimation is the use of stochastic approximation methods and stochastic gradients. We have already seen this in one setting in section 5.3.1. We review the idea again here, but in a different context. We begin with the same stochastic optimization problem, which we originally introduced as the problem

$$\min_x \mathbb{E} F(x, W).$$

Now assume that we are choosing a scalar value v to solve the problem

$$\min_v \mathbb{E} F(v, \hat{V}), \tag{17.18}$$

where

$$F(v, \hat{V}) = \frac{1}{2}(v - \hat{V})^2,$$

and where $\hat{V}$ is a random variable with unknown mean. We would like to use a series of sample realizations $\hat{v}^n$ to guide an algorithm that generates a sequence v^n that converges to the optimal solution v^* that solves (17.18). We use the same basic strategy as we introduced in section 5.3.1 where we update v^n using

$$\begin{aligned} v^n &= v^{n-1} - \alpha_{n-1} \nabla F(v^{n-1}, \hat{v}^n) \qquad (17.19) \\ &= v^{n-1} - \alpha_{n-1}(v^{n-1} - \hat{v}^n). \end{aligned}$$

Now if we make the transition that instead of updating a scalar v^n, we are updating $\overline{V}_t^n(S_t^n)$. This produces the updating equation

$$\overline{V}_t^n(S_t^n) = \overline{V}_t^{n-1}(S_t^n) - \alpha_{n-1}(\overline{V}_t^{n-1}(S_t^n) - \hat{v}^n). \tag{17.20}$$

If we use $\hat{v}^n = C(S_t^n, a_t^n) + \gamma \overline{V}^{n-1}(S_{t+1}^n)$, we quickly see that the updating equation produced using our stochastic gradient algorithm (17.20) gives us the same update that we obtained using temporal difference learning (equation (17.10)) and approximate value iteration (equation (17.11)). In equation (17.19), α_n is called a stepsize, because it controls how far we go in the direction of $\nabla F(v^{n-1}, \hat{v}^n)$, and for this reason this is the term that we adopt for α_n throughout this book. In contrast to our first use of this idea in section 5.3, where the stepsize had to serve a scaling function, in this setting the units of the variable being optimized, v^n, and the units of the gradient are the same. Indeed, we can expect that $0 < \alpha_n \leq 1$, which is a major simplification.

Now consider what happens when we replace the lookup table representation $\overline{V}(s)$ that we used above, with a linear regression $\overline{V}(s|\theta) = \theta^T \phi$. Now we want to find the best value of θ, which we can do by solving

$$\min_\theta \mathbb{E}\frac{1}{2}(\overline{V}(s|\theta) - \hat{v})^2.$$

Applying a stochastic gradient algorithm, we obtain the updating step

$$\theta^n \quad = \quad \theta^{n-1} - \alpha_{n-1}(\overline{V}(s|\theta^{n-1}) - \hat{v}^n)\nabla_\theta \overline{V}(s|\theta^n). \tag{17.21}$$

Since $\overline{V}(s|\theta^n) = \sum_{f\in\mathcal{F}} \theta^n_f \phi_f(s) = (\theta^n)^T\phi(s)$, the gradient with respect to θ is given by

$$\nabla_\theta \overline{V}(s|\theta^n) = \begin{pmatrix} \frac{\partial \overline{V}(s|\theta^n)}{\partial\theta_1} \\ \frac{\partial \overline{V}(s|\theta^n)}{\partial\theta_2} \\ \vdots \\ \frac{\partial \overline{V}(s|\theta^n)}{\partial\theta_F} \end{pmatrix} = \begin{pmatrix} \phi_1(s^n) \\ \phi_2(s^n) \\ \vdots \\ \phi_F(s^n) \end{pmatrix} = \phi(s^n).$$

Thus, the updating equation (17.21) is given by

$$\begin{aligned} \theta^n \quad &= \quad \theta^{n-1} - \alpha_{n-1}(\overline{V}(s|\theta^{n-1}) - \hat{v}^n)\phi(s^n) \\ &= \quad \theta^{n-1} - \alpha_{n-1}(\overline{V}(s|\theta^{n-1}) - \hat{v}^n) \begin{pmatrix} \phi_1(s^n) \\ \phi_2(s^n) \\ \vdots \\ \phi_F(s^n) \end{pmatrix}. \end{aligned} \tag{17.22}$$

Using a stochastic gradient algorithm requires that we have some starting estimate θ^0 for the parameter vector, although $\theta^0 = 0$ is a common choice.

While this is a simple and elegant algorithm, we have reintroduced the problem of scaling. Just as we encountered in section 5.3, the units of θ^{n-1} and the units of $(\overline{V}(s|\theta^{n-1}) - \hat{v}^n)\phi(s^n)$ may be completely different. What we have learned about stepsizes still applies, except that we may need an initial stepsize that is quite different than 1.0 (our common starting point). Our experimental work has suggested that the following policy works well: When you choose a stepsize formula, scale the first value of the stepsize so that the change in θ^n in the early iterations of the algorithm is approximately 20 to 50 percent (you will typically need to observe several iterations). You want to see individual elements of θ^n moving consistently in the same direction during the early iterations. If the stepsize is too large, the values can swing wildly, and the algorithm may not converge at all. If the changes are too small, the algorithm may simply stall out. It is very tempting to run the algorithm for a period of time and then conclude that it appears to have converged (presumably to a good solution). While it is important to see the individual elements moving in the same direction (consistently increasing or decreasing) in the early iterations, it is also important to see oscillatory behavior toward the end.

17.3 BELLMAN'S EQUATION USING A LINEAR MODEL

It is possible to solve Bellman's equation for infinite horizon problems by starting with the assumption that the value function is given by a linear model $V(s) = \theta^T\phi(s)$ where $\Phi(s)$

is a column vector of basis functions for a particular state s. Of course, we are still working with a single policy, so we are using Bellman's equation only as a method for finding the best linear approximation for the infinite horizon value of a fixed policy π.

We begin with a derivation based on matrix linear algebra, which is more advanced and which does not produce expressions that can be implemented in practice. We follow this discussion with a simulation-based algorithm which can be implemented fairly easily.

17.3.1 A matrix-based derivation*

In section 18.4.2, we provided a geometric view of basis functions, drawing on the elegance and obscurity of matrix linear algebra. We are going to continue this presentation and present a version of Bellman's equation assuming linear models. However, we are not yet ready to introduce the dimension of optimizing over policies, so we are still simply trying to approximate the value of being in a state. Also, we are only considering infinite horizon models, since we have already handled the finite horizon case. This presentation can be viewed as another method for handling infinite horizon models, while using a linear architecture to approximate the value function.

First recall that Bellman's equation (for a fixed policy) is written

$$V^\pi(s) = C(s, A^\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, A^\pi(s)) V^\pi(s').$$

In vector-matrix form, we let V^π be a vector with element $V^\pi(s)$, we let c^π be a vector with element $C(s, A^\pi(s))$ and finally we let P^π be the one-step transition matrix with element $p(s'|s, A^\pi(s))$ at row s, column s'. Using this notation, Bellman's equation becomes

$$V^\pi = c^\pi + \gamma P^\pi V^\pi,$$

allowing us to solve for V^π using

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

This works with a lookup-table representation (a value for each state). Now assume that we replace V^π with an approximation $\overline{V}^\pi = \Phi\theta$ where, Φ is a $|\mathcal{S}| \times |\mathcal{F}|$ matrix with element $\Phi_{s,f} = \phi_f(s)$. Also let d_s^π be the steady state probability of being in state s while following policy π, and let D^π be a $|\mathcal{S}| \times |\mathcal{S}|$ diagonal matrix where the state probabilities $(d_1^\pi, \ldots, d_{|\mathcal{S}|}^\pi$ make up the diagonal. We would like to choose θ to minimize the weighted sum of errors squared, where the error for state s is given by

$$\epsilon^n(s) = \sum_f \theta_f \phi_f(s) - \left(c^\pi(s) + \gamma \sum_{s' \in \mathcal{S}} p^\pi(s'|s, A^\pi) \sum_f \theta_f^n \phi_f(s') \right). \qquad (17.23)$$

The first term on the right hand side of (17.23) is the predicted value of being in each state given θ, while the second term on the right hand side is the "predicted" value computed using the one-period contribution plus the expected value of the future which is computed using θ^n. The expected sum of errors squared is then given by

$$\min_\theta \sum_{s \in \mathcal{S}} d_s^\pi \left(\sum_f \theta_f \phi_f(s) - \left(c^\pi(s) + \gamma \sum_{s' \in \mathcal{S}} p^\pi(s'|s, A^\pi) \sum_f \theta_f^n \phi_f(s') \right) \right)^2,$$

In matrix form, this can be written

$$\min_{\theta}(\Phi\theta - (c^{\pi} + \gamma P^{\pi}\Phi\theta^{n}))^{T} D^{\pi}(\Phi\theta - (c^{\pi} + \gamma P^{\pi}\Phi\theta^{n})) \tag{17.24}$$

where D^{π} is a $|\mathcal{S}| \times |\mathcal{S}|$ diagonal matrix with elements d_s^{π} which serves a scaling role (we want to focus our attention on states we visit the most). We can find the optimal value of θ (given θ^{n}) by taking the derivative of the function being optimized in (17.24) with respect to θ and setting it equal to zero. Let θ^{n+1} be the optimal solution, which means we can write

$$\Phi^{T} D^{\pi}\big(\Phi\theta^{n+1} - (c^{\pi} + \gamma P^{\pi}\Phi\theta^{n})\big) = 0, \tag{17.25}$$

We can find a fixed point $\lim_{n\to\infty}\theta^{n} = \lim_{n\to\infty}\theta^{n+1} = \theta^{*}$, which allows us to write equation (17.25) in the form

$$A\theta^{*} = b, \tag{17.26}$$

where $A = \Phi^{T} D^{\pi}(I - \gamma P^{\pi})\Phi$ and $b = \Phi^{T} D^{\pi} c^{\pi}$. This allows us, in theory at least, to solve for θ^{*} using

$$\theta^{*} = A^{-1}b, \tag{17.27}$$

which can be viewed as a scaled version of the normal equations (equation 3.56). Equation (17.27) is very similar to our calculation of the steady state value of being in each state introduced in chapter 14, given by

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

Equation (17.27) differs only in the scaling by the probability of being in each state (D^{π}) and then the transformation to the feature space by Φ.

We note that equation (17.25) can also be written in the form

$$A\theta - b = \Phi^{T} D^{\pi}\left(\Phi\theta - (c^{\pi} + \gamma P^{\pi}\Phi\theta)\right). \tag{17.28}$$

The term $\Phi\theta$ can be viewed as the approximate value of each state. The term $(c^{\pi} + \gamma P^{\pi}\Phi\theta)$ can be viewed as the one-period contribution plus the expected value of the state that you transition to under policy π, again computed for each state. Let δ^{π} be a column vector containing the temporal difference for each state when we choose an action according to policy π. By tradition, the temporal difference has always been written in the form $C(S_t, a) + \overline{V}(S_{t+1}) - \overline{V}(S_t)$, which can be thought of as "estimated minus predicted." If we continue to let δ^{π} be the traditional definition of the temporal difference, it would be written

$$\delta^{\pi} = -\big(\Phi\theta - (c^{\pi} + \gamma P^{\pi}\Phi\theta)\big). \tag{17.29}$$

The pre-multiplication of δ^{π} by D^{π} in (17.28) has the effect of factoring each temporal difference by the probability that we are in each state. Then pre-multiplying $D^{\pi}\delta^{\pi}$ by Φ^{T} has the effect of transforming this scaled temporal difference for each state into the feature space.

The goal is to find the value θ that produces $A\theta - b = 0$, which means we are trying to find the value θ that produces a scaled version of $\Phi\theta - (c^{\pi} + \gamma P^{\pi}\Phi\theta) = 0$, but transformed to the feature space.

Linear algebra offers a compact elegance, but at the same time can be hard to parse, and for this reason we encourage the reader to stop and think about the relationships. One useful exercise is to think of a set of basis functions where we have a "feature" for each state, with $\phi_f(s) = 1$ if feature f corresponds to state s. In this case, Φ is the identity matrix. D^π, the diagonal matrix with diagonal elements d_s^π giving the probability of being in state s, can be viewed as scaling quantities for each state by the probability of being in a state. If Φ is the identity matrix, then $A = D^\pi - \gamma D^\pi P^\pi$ where $D^\pi P^\pi$ is the matrix of *joint* probabilities of being in state s *and* then transitioning to state s'. The vector b becomes the vector of the cost of being in each state (and then taking the action corresponding to policy π) times the probability of being in the state.

When we have a smaller set of basis functions, then multiplying c^π or $D^\pi(I - \gamma P^\pi)$ times Φ has the effect of scaling quantities that are indexed by the state into the feature space, which also transforms an $|\mathcal{S}|$-dimensional space into an $|\mathcal{F}|$-dimensional space.

17.3.2 A simulation-based implementation

We start by simulating a trajectory of states, actions and information,

$$(S^0, a^0, W^1, S^1, a^1, W^2, \ldots, S^n, a^n, W^{n+1}).$$

Recall that $\phi(s)$ is a column vector with an element $\phi_f(s)$ for each feature $f \in \mathcal{F}$. Using our simulation above, we also obtain a sequence of column vectors $\phi(s^i)$ and contributions $C(S^i, a^i, W^{i+1})$. We can create a sample estimate of the $|\mathcal{F}|$ by $|\mathcal{F}|$ matrix A in the section above using

$$A^n = \frac{1}{n}\sum_{i=0}^{n-1} \phi(S^i)(\phi(S^i) - \gamma\phi(S^{i+1}))^T. \qquad (17.30)$$

We can also create a sample estimate of the vector b using

$$b^n = \frac{1}{n}\sum_{i=0}^{n-1} \phi(S^i)C(S^i, a^i, W^{i+1}). \qquad (17.31)$$

To gain some intuition, again stop and assume that there is a feature for every state, which means that $\phi(S^i)$ is a vector of 0's with a 1 corresponding to the element for state i, which means it is a kind of indicator variable telling us what state we are in. The term $(\phi(S^i) - \gamma\phi(S^{i+1}))$ is then a simulated version of $D^\pi(I - \gamma P^\pi)$, weighted by the probability that we are in a particular state, where we replace the probability of being in a state with a sampled realization of actually being in a particular state.

We are going to use this foundation to introduce two important algorithms for infinite horizon problems when using linear models to approximate value function approximations. These are known as *least squares temporal differences* (LSTD), and *least squares policy evaluation* (LSPE).

17.3.3 Least squares temporal differences (LSTD)

As long as A^n is invertible (which is not guaranteed), we can compute a sample estimate of θ using

$$\theta^n = (A^n)^{-1} b^n. \qquad (17.32)$$

This algorithm is known in the literature as *least squares temporal differences*. As long as the number of features is not too large (as is typically the case), the inverse is not too hard to compute. LSTD can be viewed as a batch algorithm which operates by collecting a sample of temporal differences, and then using least squares regression to find the best linear fit.

We can see the role of temporal differences more clearly by doing a little algebra. We use equations (17.30) and (17.31) to write

$$\begin{aligned} A^n\theta^n - b^n &= \frac{1}{n}\sum_{i=0}^{n-1}\left(\phi(S^i)(\phi(S^i) - \gamma\phi(S^{i+1}))^T\theta^n - \phi(S^i)C(S^i,a^i,W^{i+1})\right) \\ &= \frac{1}{n}\sum_{i=0}^{n-1}\phi(S^i)\left(\phi(S^i)^T\theta^n - (c^\pi + \alpha\phi(S^{i+1})^T\theta^n)\right) \\ &= \frac{1}{n}\sum_{i=0}^{n-1}\phi(S^i)\delta^i(\theta^n), \end{aligned}$$

where $\delta^i(\theta^n) = \phi(S^i)^T\theta^n - (c^\pi + \alpha\phi(S^{i+1})^T\theta^n)$ is the ith temporal difference given the parameter vector θ^n. Thus, we are doing a least squares regression so that the sum of the temporal differences over the simulation (which approximations the expectation) is equal to zero. We would, of course, like it if θ could be chosen so that $\delta^i(\theta) = 0$ for all i. However, when working with sample realizations the best we can expect is that the average across the observations of $\delta^i(\theta)$ tends to zero.

17.3.4 Least squares policy evaluation (LSPE)

LSTD is basically a batch algorithm, which requires collecting a sample of n observations and then using regression to fit a model. An alternative strategy uses a stochastic gradient algorithm which successively updates estimates of θ. The basic updating equation is

$$\theta^n = \theta^{n-1} - \frac{\alpha}{n}G^n\sum_{i=0}^{n-1}\phi(S^i)\delta^i(n), \tag{17.33}$$

where G^n is a scaling matrix. Although there are different strategies for computing G^n, the most natural is a simulation-based estimate of $(\Phi^T D^\pi\Phi)^{-1}$ which can be computed using

$$G^n = \left(\frac{1}{n+1}\sum_{i=0}^{n}\phi(S^i)\phi(S^i)^T\right)^{-1}.$$

To visualize G^n, return again to the assumption that there is a feature for every state. In this case, $\phi(S^i)\phi(S^i)^T$ is an $|\mathcal{S}|$ by $|\mathcal{S}|$ matrix with a 1 on the diagonal for row S^i and column S^i. As n approaches infinity, the matrix

$$\left(\frac{1}{n+1}\sum_{i=0}^{n}\phi(S^i)\phi(S^i)^T\right)$$

approaches the matrix D^π of the probability of visiting each state, stored in elements along the diagonal.

17.4 ANALYSIS OF TD(0), LSTD AND LSPE USING A SINGLE STATE

A useful exercise to understand the behavior of recursive least squares, LSTD and LSPE is to consider what happens when they are applied to a trivial dynamic program with a single state and a single action. Obviously, we are interested in the policy that chooses the single action. This dynamic program is equivalent to computing the sum

$$F = \mathbb{E}\sum_{i=0}^{\infty} \gamma^i \hat{C}^i, \tag{17.34}$$

where $\hat{C}^i$ is a random variable giving the ith contribution. If we let $\bar{c} = \mathbb{E}\hat{C}^i$, then clearly $F = \frac{1}{1-\gamma}\bar{c}$. But let's pretend that we do not know this, and we are using these various algorithms to compute the expectation.

17.4.1 Recursive least squares and TD(0)

Let $\hat{v}^n$ be an estimate of the value of being in state S^n. We continue to assume that the value function is approximated using

$$\overline{V}(s) = \sum_{f\in\mathcal{F}} \theta_f \phi_f(s).$$

We wish to choose θ by solving

$$\min_{\theta} \sum_{i=1}^{n} \left(\hat{v}^i - \left(\sum_{f\in\mathcal{F}} \theta_f \phi_f(S^i) \right) \right)^2 .$$

Let θ^n be the optimal solution. We can determine this recursively using the techniques presented earlier in this chapter which gives us the updating equation

$$\theta^n = \theta^{n-1} - \frac{1}{1+(x^n)^T B^{n-1} x^n} B^{n-1} x^n (\overline{V}^{n-1}(S^n) - \hat{v}^n) \tag{17.35}$$

where $x^n = (\phi_1(S^n), \ldots, \phi_f(S^n), \ldots, \phi_F(S^n))$, and the matrix B^n is computed using

$$B^n = B^{n-1} - \frac{1}{1+(x^n)^T B^{n-1} x^n} \left(B^{n-1} x^n (x^n)^T B^{n-1} \right).$$

If we have only one state and one action, we only have one basis function $\phi(s) = 1$ and one parameter $\theta^n = \overline{V}^n(s)$. Now the matrix B^n is a scalar and equation (17.35) reduces to

$$\begin{aligned} v^n &= v^{n-1} - \frac{B^{n-1}}{1+B^{n-1}}(v^{n-1} - \hat{v}^n) \\ &= \left(1 - \frac{B^{n-1}}{1+B^{n-1}}\right) v^{n-1} + \frac{B^{n-1}}{1+B^{n-1}}. \end{aligned}$$

If $B^0 = 1$, $B^{n-1} = 1/n$, giving us

$$v^n = \frac{n-1}{n} v^{n-1} + \frac{1}{n}\hat{v}^n.$$

Imagine now that we are using TD(0) where $\hat{v}^n = \hat{C}^n + \gamma v^{n-1}$. In this case, we obtain

$$v^n = \left(1 - (1-\gamma)\frac{1}{n}\right) v^{n-1} + \frac{1}{n}\hat{C}^n. \tag{17.36}$$

Equation (17.36) can be viewed as an algorithm for finding

$$v = \sum_{n=0}^{\infty} \gamma^n \hat{C}^n,$$

where the solution is $v^* = \frac{1}{1-\gamma}\mathbb{E}\hat{C}$.

Equation (17.36) shows us that recursive least squares, when $\hat{v}^n$ is computed using temporal difference learning, has the effect of successively adding sample realizations of costs, with a "discount factor" of $1/n$. The factor $1/n$ arises directly as a result of the need to smooth out the noise in $\hat{C}^n$. For example, if $\hat{C} = c$ is a known constant, we could use standard value iteration, which would give us

$$v^n = c + \gamma v^{n-1}. \tag{17.37}$$

It is easy to see that v^n in (17.37) will rise much more quickly toward v^* than the algorithm in equation (17.36). We return to this topic in some depth in chapter 6.

17.4.2 LSPE

LSPE requires that we first generate a sequence of states S^i and contributions $\hat{C}^i$ for $i = 1, \ldots, n$. We then compute θ by solving the regression problem

$$\theta^n = \arg\min_{\theta} \sum_{i=1}^{n} \left(\sum_f \theta_f \phi_f(S^i) - \left(\hat{C}^i + \gamma \overline{V}^{n-1}(S^{i+1})\right)\right)^2.$$

For a problem with one state where $\theta^n = v^n$, this reduces to

$$v^n = \arg\min_{\theta} \sum_{i=1}^{n} \left(\theta - \left(\hat{C}^i + \gamma v^{n-1}\right)\right)^2.$$

This problem can be solved in closed form, giving us

$$v^n = \left(\frac{1}{n}\sum_{i=1}^{n} \hat{C}^i\right) + \gamma v^{n-1}.$$

17.4.3 LSTD

Finally, we showed above that the LSTD procedure finds θ by solving the system of equations

$$\sum_{i=1}^{n} \phi_f(S^i)(\phi_f(S^i) - \gamma\phi_f(S^{i+1}))^T \theta^n = \sum_{i=1}^{n} \phi_f(S^i)\hat{C}^i,$$

for each $f \in \mathcal{F}$. Again, since we have only one basis function $\phi(s) = 1$ for our single state problem, this reduces to finding the scalar $\theta^n = v^n$ using

$$v^n = \frac{1}{1-\gamma}\left(\frac{1}{n}\sum_{i=1}^{n} \hat{C}^n\right).$$

17.4.4 Discussion

This presentation illustrates three different styles for estimating an infinite horizon sum. In recursive least squares, equation (17.35) demonstrates the successive smoothing of the previous estimate v^n and the latest estimate $\hat{v}^n$. We are, at the same time, adding contributions over time while also trying to smooth out the noise.

LSPE, by contrast, separates the estimation of the mean of the single period contribution, and the process of summing contributions over time. At each iteration, we improve our estimate of $\mathbb{E}\hat{C}$, and then accumulate our latest estimate in a telescoping sum.

LSTD, finally, updates its estimate of $\mathbb{E}\hat{C}$, and then projects this over the infinite horizon by factoring the result by $1/(1-\gamma)$.

17.5 GRADIENT-BASED METHODS FOR APPROXIMATE VALUE ITERATION

There has been a strong desire for approximation algorithms with the following features: 1) off-policy learning, 2) temporal-difference learning, 3) linear models for value function approximation and 4) complexity (in memory and computation) that is linear in the number of features. The last requirement is primarily of interest in specialized applications which require thousands or even millions of features. Off-policy learning is desirable because it provides an important degree of control over exploration. Temporal-difference learning is useful because it is so simple, as are the use of linear models, which make it possible to provide an estimate of the entire value function with a small number of measurements.

Off-policy, temporal-difference learning was first introduced in the form of Q-learning using a lookup table representation, where it is known to convergence. But we lose this property if we introduce value function approximations that are linear in the parameters. In fact, Q-learning can be shown to diverge for any positive stepsize. The reason is that there is no guarantee that our linear model is accurate, which can introduce significant instabilities in the learning process.

Q-learning and temporal difference learning can be viewed as forms of stochastic gradient algorithms, but the problem with earlier algorithms when we use linear value function approximations can be traced to the choice of objective function. For example, if we wish to find the best linear approximation $\overline{V}(s|\theta)$, a hypothetical objective function would be to minimize the expected mean squared difference between $\overline{V}(s|\theta)$ and the true value function $V(s)$. If d_s^π is the probability of being in state s, this objective would be written

$$MSE(\theta) = \frac{1}{2}\sum_s d_s^\pi (\overline{V}(s|\theta) - V(s))^2.$$

If we are using approximate value iteration, a more natural objective function is to minimize the mean squared Bellman error. We use the Bellman operator $\mathcal{M}^\pi$ (as we did in chapter 14) for policy π to represent

$$\mathcal{M}^\pi v = c^\pi + \gamma P^\pi v,$$

where v is a column vector giving the value of being in state s, and c^π is the column vector of contributions $C(s, A^\pi(s))$ if we are in state s and choose an action a according to policy π. This allows us to define

$$\begin{aligned} MSBE(\theta) &= \frac{1}{2}\sum_s d_s^\pi \left(\overline{V}(s|\theta) - (c^\pi(s) + \gamma \sum_{s'} p^\pi(s'|s)\overline{V}(s'|\theta))\right)^2 \\ &= \|\overline{V}(\theta) - \mathcal{M}\overline{V}(\theta)\|_D^2. \end{aligned}$$

We can minimize MSBE(θ) by generating a sequence of states $(S^1, \ldots, S^i, S^{i+1}, \ldots)$ and then computing a stochastic gradient

$$\nabla_\theta \text{MSBE}(\theta) \quad = \quad \delta^{\pi,i}(\phi(S^i) - \gamma\phi(S^{i+1}))$$

where $\phi(S^i)$ is a column vector of basis functions evaluated at state S^i. The scalar $\delta^{\pi,i}$ is the temporal difference given by

$$\delta^{\pi,i} = \overline{V}(S^i|\theta) - (c^\pi(S^i) + \gamma\overline{V}(S^{i+1}|\theta)).$$

We note that $\delta^{\pi,i}$ depends on the policy π which affects both the single period contribution and the likelihood of transitioning to state S^{i+1}. To emphasize that we are working with a fixed policy, we carry the superscript π throughout.

For this section, we are defining the temporal difference as $\delta^{\pi,i} = \overline{V}(S^i|\theta) - (c^\pi(S^i) + \gamma\overline{V}(S^{i+1}|\theta))$, because it is a natural byproduct when deriving algorithms based on stochastic gradient methods. Earlier in this chapter, we defined the temporal difference as $\delta_\tau = C(S^n_\tau, a^n_\tau, W^n_{\tau+1}) + \overline{V}^{n-1}_{\tau+1}(S^n_{\tau+1}) - \overline{V}^{n-1}_\tau(S^n_\tau)$ (see equation (17.4)), which is more natural when used to represent telescoping sums (for example, see equation (17.5)). A stochastic gradient algorithm, then, would seek to optimize θ using

$$\begin{aligned} \theta^{n+1} &= \theta^n - \alpha_n \nabla_\theta \text{MSBE}(\theta) && (17.38) \\ &= \theta^n - \alpha_n \delta^{\pi,n}(\phi(S^n) - \gamma\phi(S^{n+1})). && (17.39) \end{aligned}$$

Were we to use the more traditional definition of a temporal difference, our equation would be written

$$\theta^{n+1} = \theta^n + \alpha_n \delta^{\pi,n}(\phi(S^n) - \gamma\phi(S^{n+1})),$$

which runs counter to the classical statement of a stochastic gradient algorithm (given in equation (17.38)) for minimization problems.

A variant of this basic algorithm, called the generalized TD(0) (or, GTD(0)) algorithm, is given by

$$\theta^{n+1} \quad = \quad \theta^n - \alpha_n(\phi(S^n) - \gamma\phi(S^{n+1}))\phi(S^n)^T u^n, \qquad (17.40)$$

where

$$u^{n+1} \quad = \quad u^n - \beta_n(u^n - \delta^{\pi,n}\phi(S^n)). \qquad (17.41)$$

α_n and β_n are both stepsizes. u^n is a smoothed estimate of the product $\delta^{\pi,n}\phi(S^n)$.

Gradient descent methods based on temporal differences will not minimize MSBE(θ) because there does not exist a value of θ that would allow $\hat{v}(s) = c^\pi(s) + \gamma\overline{V}(s|\theta)$ to be represented as $\overline{V}(s|\theta)$. We can fix this using the mean squared projected Bellman error (MSPBE(θ)) which we compute as follows. It is more compact to do this development using matrix-vector notation. We first recall the projection operator Π given by

$$\Pi = \Phi(\Phi^T D^\pi \Phi)^{-1}\Phi^T D^\pi.$$

(See section 18.4.2 for a derivation of this operator.) If V is a vector giving the value of being in each state, ΠV is the nearest projection of V on the space generated by $\theta\phi(s)$. We are trying to find $\overline{V}(\theta)$ that will match the one-step lookahead given by $\mathcal{M}^\pi\overline{V}(\theta)$, but this produces a column vector that cannot be represented directly as $\Phi\theta$, where Φ is the

$|\mathcal{S}| \times |\mathcal{F}|$ matrix of feature vectors ϕ. We accomplish this by pre-multiplying $\mathcal{M}^\pi V(\theta)$ by the projection operator Π. This allows us to form the mean squared projected Bellman error using

$$\begin{aligned} MSPBE(\theta) &= \frac{1}{2}\|\overline{V}(\theta) - \Pi\mathcal{M}^\pi\overline{V}(\theta)\|_D^2 && (17.42) \\ &= \frac{1}{2}\big(\overline{V}(\theta) - \Pi\mathcal{M}^\pi\overline{V}(\theta)\big)^T D\big(\overline{V}(\theta) - \Pi\mathcal{M}^\pi\overline{V}(\theta)\big). && (17.43) \end{aligned}$$

We can now use this new objective function as the basis of an optimization algorithm to find θ. Recall that D^π is a $|\mathcal{S}| \times |\mathcal{S}|$ diagonal matrix with elements d_s^π, giving us the probability that we are in state s while following policy π. We use D^π as a scaling matrix to give us the probability that we are in state s. We start by noting the identities

$$\begin{aligned} \mathbb{E}[\phi\phi^T] &= \sum_{s\in\mathcal{S}} d_s^\pi \phi_s \phi_s^T \\ &= \Phi^T D^\pi \Phi. \\ \mathbb{E}[\delta^\pi\phi] &= \sum_{s\in\mathcal{S}} d_s^\pi \phi_s \left(c^\pi(s) + \gamma \sum_{s'\in\mathcal{S}} p^\pi(s'|s)\overline{V}(s'|\theta) - \overline{V}(s|\theta) \right) \\ &= \Phi^T D^\pi(\mathcal{M}^\pi\overline{V}(\theta) - \overline{V}(\theta)). \end{aligned}$$

The derivations here and below make extensive use of matrices, which can be difficult to parse. A useful exercise is to write out the matrices assuming that there is a feature $\phi_f(s)$ for each state s, so that $\phi_f(s) = 1$ if feature f corresponds to state s. See exercise 17.3.

We see that the role of the scaling matrix D^π is to enable us to take the expectation of the quantities $\phi\phi^T$ and $\delta^\pi\phi$. Below, we are going to simulate these quantities, where a state will occur with probability d_s^π. We also use

$$\begin{aligned} \Pi^T D^\pi \Pi &= (\Phi(\Phi^T D^\pi\Phi)^{-1}\Phi^T D^\pi)^T D^\pi(\Phi(\Phi^T D^\pi\Phi)^{-1}\Phi^T D^\pi) \\ &= (D^\pi)^T\Phi(\Phi^T D^\pi\Phi)^{-1}\Phi^T D^\pi\Phi(\Phi^T D^\pi\Phi)^{-1}\Phi^T D^\pi \\ &= (D^\pi)^T\Phi(\Phi^T D^\pi\Phi)^{-1}\Phi^T D^\pi. \end{aligned}$$

We have one last painful piece of linear algebra that gives us a more compact form for MSPBE(θ). Pulling the $1/2$ to the left hand side (this will later vanish when we take the derivative), we can write

$$\begin{aligned} 2MSPBE(\theta) &= \|\overline{V}(\theta) - \Pi\mathcal{M}^\pi\overline{V}(\theta)\|_D^2 \\ &= \|\Pi(\overline{V}(\theta) - \mathcal{M}^\pi\overline{V}(\theta))\|_D^2 \\ &= (\Pi(\overline{V}(\theta) - \mathcal{M}^\pi\overline{V}(\theta)))^T D^\pi(\Pi(\overline{V}(\theta) - \mathcal{M}^\pi\overline{V}(\theta))) \\ &= (\overline{V}(\theta) - \mathcal{M}^\pi\overline{V}(\theta))^T\Pi^T D^\pi\Pi(\overline{V}(\theta) - \mathcal{M}^\pi\overline{V}(\theta)) \\ &= (\overline{V}(\theta) - \mathcal{M}^\pi\overline{V}(\theta))^T(D^\pi)^T\Phi(\Phi^T(D^\pi)\Phi)^{-1}\Phi^T D^\pi(\overline{V}(\theta) - \mathcal{M}^\pi\overline{V}(\theta)) \\ &= (\Phi^T D^\pi(\mathcal{M}^\pi\overline{V}(\theta) - \overline{V}(\theta)))^T(\Phi^T D^\pi\Phi)^{-1}\Phi^T D^\pi(\mathcal{M}\overline{V}(\theta) - \overline{V}(\theta)) \\ &= \mathbb{E}[\delta^\pi\phi]^T\mathbb{E}[\phi\phi^T]^{-1}\mathbb{E}[\delta^\pi\phi]. && (17.44) \end{aligned}$$

We next need to estimate the gradient of this error $\nabla_\theta MSPBE(\theta)$. Keep in mind that $\delta^\pi = c^\pi + \gamma P^\pi\Phi\theta - \Phi\theta$. If ϕ is the column vector with element $\phi(s)$, assume that s' occurs with probability $p^\pi(s'|s)$ under policy π, and let ϕ' be the corresponding column vector. Differentiating (17.44) gives

$$\begin{aligned}\nabla_\theta MSPBE(\theta) &= \mathbb{E}[(\gamma\phi' - \phi)\phi^T]\mathbb{E}[\phi\phi^T]^{-1}\mathbb{E}[\delta^\pi\phi] \\ &= -\mathbb{E}[(\phi - \gamma\phi')\phi^T]\mathbb{E}[\phi\phi^T]^{-1}\mathbb{E}[\delta^\pi\phi].\end{aligned}$$

We are going to use a standard stochastic gradient updating algorithm for minimizing the error given by $MSPBE(\theta)$, which is given by

$$\begin{aligned}\theta^{n+1} &= \theta^n - \alpha_n \nabla_\theta MSPBE(\theta) &(17.45)\\ &= \theta^n + \alpha_n\mathbb{E}[(\phi - \gamma\phi')\phi^T]\mathbb{E}[\phi\phi^T]^{-1}\mathbb{E}[\delta^\pi\phi]. &(17.46)\end{aligned}$$

We can create a linear predictor which approximates

$$w \approx \mathbb{E}[\phi\phi^T]^{-1}\mathbb{E}[\delta^\pi\phi].$$

where w is approximated using

$$w^{n+1} = w^n + \beta_n(\delta^{\pi,n} - (\phi^n)^T w^n)\phi^n.$$

This allows us to write the gradient

$$\begin{aligned}\nabla_\theta MSPBE(\theta) &= -\mathbb{E}[(\phi - \gamma\phi')\phi^T]\mathbb{E}[\phi\phi^T]^{-1}\mathbb{E}[\delta^\pi\phi] \\ &\approx -\mathbb{E}[(\phi - \gamma\phi')\phi^T]w.\end{aligned}$$

We have now created the basis for two algorithms. The first is called generalized temporal difference 2 (GTD2), given by

$$\theta^{n+1} = \theta^n + \alpha_n(\phi^n - \gamma\phi^{n+1})((\phi^n)^T w^n). \qquad (17.47)$$

Here, ϕ^n is the column vector of basis functions when we are in state S^n, while ϕ^{n+1} is the column vector of basis functions for the next state S^{n+1}. Note that if equation (17.47) is executed right to left, all calculations are linear in the number of features F.

An important feature of the algorithm, especially for applications with large number of features, is that the algorithm is linear in the number of features.

A variant, called TDC (temporal difference with gradient corrector) is derived by using a slightly modified calculation of the gradient

$$\begin{aligned}\nabla_\theta MSPBE(\theta) &= -\mathbb{E}[(\phi - \gamma\phi')\phi^T]\mathbb{E}[\phi\phi^T]^{-1}\mathbb{E}[\delta^\pi\phi] \\ &= -\left(\mathbb{E}[\phi\phi^T] - \gamma\mathbb{E}[\phi'\phi^T]\right)\mathbb{E}[\phi\phi^T]^{-1}\mathbb{E}[\delta^\pi\phi] \\ &= -\left(\mathbb{E}[\delta^\pi\phi] - \gamma\mathbb{E}[\phi'\phi^T]\mathbb{E}[\phi\phi^T]^{-1}\mathbb{E}[\delta^\pi\phi]\right) \\ &\approx -\left(\mathbb{E}[\delta^\pi\phi] - \gamma\mathbb{E}[\phi'\phi^T]w\right).\end{aligned}$$

This gives us the TDC algorithm

$$\theta^{n+1} = \theta^n + \alpha_n\left(\delta^{\pi,n}\phi^n - \gamma\phi^{n'}((\phi^n)^T w^n)\right). \qquad (17.48)$$

GTD2 and TDC are both proven to converge to the optimal value of θ for a fixed target policy $A^\pi(s)$ which may be different than the behavior (sampling) policy. That is, after updating θ^n where the temporal difference $\delta^{\pi,n}$ is computed assuming we are in state S^n and follow policy π, we are allowed to follow a separate behavior policy to determine S^{n+1}. This allows us to directly control the states that we visit, rather than depending on the decisions made by the target policy.

17.6 LEAST SQUARES TEMPORAL DIFFERENCING WITH KERNEL REGRESSION*

In section 3.10.2, we introduced the idea of kernel regression, where we can approximate the value of a function by using a weighted sum of nearby observations. If S^i is the ith observation of a state, and we observe a value $\hat{v}^i$, we can approximate the value of visiting a generic state s by using

$$\overline{V}(s) = \frac{\sum_{i=1}^n K_h(s, S^i)\hat{v}^i}{\sum_{i=1}^n K_h(s, S^i)},$$

where $K_h(s, S^i)$ is a weighting function that declines with the distance between s and S^i. Kernel functions are introduced in section 3.10.2.

We now use two properties of kernels. One, which we first introduced in section 3.10.2, is that for most kernel functions $K_h(s, s')$, there exists a potentially high-dimensional set of basis functions $\phi(s)$ such that

$$K_h(s, s') = \phi(s)^T \phi(s').$$

There is also a result known as the kernel representor theorem that states that there exists a vector of coefficients $\beta_i,\ i = 0, \ldots, m$ that allows us to write

$$\theta^m = \sum_{i=0}^m \phi(s^i)\beta^i. \tag{17.49}$$

This allows us to write our value function approximation using

$$\begin{aligned}\overline{V}(s) &= \phi(s)^T \theta^m \\ &= \sum_{i=0}^m \phi(S^i)^T \beta^i \\ &= \sum_{i=0}^m \phi(s)^T \phi(S^i)\beta^i \\ &= \sum_{i=0}^m K_h(s, S^i)\beta^i.\end{aligned}$$

Recall from equations (17.26), (17.30) and (17.31) that

$$A^m \theta = b^m,$$

where

$$\left(\frac{1}{m}\sum_{i=1}^m \phi(S^i)(\phi(S^i) - \gamma\phi(S^{i+1}))^T\right)\theta = \frac{1}{m}\sum_{i=1}^m \phi(S^i)\hat{C}^i + \varepsilon^i,$$

where ε^i represents the error in the fit. Substituting in (17.49) gives us

$$\left(\frac{1}{m}\sum_{i=1}^m \phi(S^i)(\phi(S^i) - \gamma\phi(S^{i+1}))^T\right)\sum_{i=1}^m \phi^i\beta^i = \frac{1}{m}\sum_{i=1}^m \phi(S^i)\hat{C}^i + \varepsilon^i.$$

The single step regression function is given by

$$\phi(S^i)(\phi(S^i) - \gamma\phi(S^{i+1}))^T \sum_{i=1}^{m} \phi^i \beta^i = \frac{1}{m} \sum_{i=1}^{m} \phi(S^i)\hat{C}^i + \varepsilon^i. \tag{17.50}$$

Let $\Phi^m = [\phi(S^1), \ldots, \phi(S^m)]^T$ be a $m \times |\mathcal{F}|$ matrix, where each row is the vector $\phi(S^i)$, and let $k^m(S^i) = [K_h(S^1, S^i), \ldots, K_h(S^m, S^i)]^T$ be a column vector of the kernel functions capturing the weight between each observed state $S^1, \ldots, S^m$, and a particular state S^i. Multiplying both sides of (17.50) by Φ^m gives us,

$$\Phi^m \phi(S^i)(\phi(S^i) - \gamma\phi(S^{i+1}))^T \sum_{i=1}^{m} \phi^i \beta^i = \Phi^m \frac{1}{m} \sum_{i=1}^{m} \phi(S^i)\hat{C}^i + \Phi^m \varepsilon^i.$$

Keeping in mind that products of basis functions can be replaced with kernel functions, we obtain

$$\sum_{i=1}^{m} k^m(S^i)(k^m(S^i) - \gamma k^m(S^{i+1}))\beta^m = \sum_{i=1}^{m} k^m(S^i)\hat{C}^i.$$

Define the $m \times m$ matrix and m-vector b^m

$$\begin{aligned} M^m &= \sum_{i=1}^{m} k^m(S^i)(k^m(s^i) - \gamma k^m(S^{i+1})), \\ b^m &= \sum_{i=1}^{m} k^m(S^i)\hat{C}^i. \end{aligned}$$

Then, we can solve for β^m recursively using

$$\begin{aligned} \beta^m &= (M^m)^{-1} b^m, \\ M^{m+1} &= M^m + k^{m+1}(S^{m+1})((k^{m+1}(S^{m+1}))^T - \gamma(k^{m+2}(S^{m+2}))^T), \\ b^{m+1} &= b^m + k^{m+1}(S^{m+1})\hat{C}^m. \end{aligned}$$

The power of kernel regression is that it does not require specifying basis functions. However, this flexibility comes with a price. If we are using a parametric model, we have to deal with a vector θ with $|\mathcal{F}|$ elements. Normally we try to specify a relatively small number of features, although there are applications which might use a million features. With kernel regression, we have to invert the $m \times m$ matrix M^m, which can become very expensive as m grows. As a result, practical algorithms would have to use advanced research on sparsification.

17.7 VALUE FUNCTION APPROXIMATIONS BASED ON BAYESIAN LEARNING*

A different strategy for updating value functions is one based on Bayesian learning. Assume that we start with a prior $V^0(s)$ of the value of being in state s, and we assume that we have a known covariance function $Cov(s, s')$ that captures the relationship in our belief about $V(s)$ and $V(s')$. A good example where this function would be known might be a function where s is continuous (or a discretization of a continuous surface), where we might use

$$Cov(s, s') \propto e^{-\frac{\|s-s'\|^2}{b}} \tag{17.51}$$

where b is a bandwidth. This function captures the intuitive behavior that if two states are close to each other, their covariance is higher. So, if we make an observation that raises our belief about $V(s)$, then our belief about $V(s')$ will increase also, and will increase more if s and s' are close to each other. We also assume that we have a variance function $\lambda(s)$ that captures the noise in a measurement $\hat{v}(s)$ of the function at state s.

Our Bayesian updating model is designed for applications where we have access to observations $\hat{v}^n$ of our true function $V(s)$ which we can view as coming from our prior distribution of belief. This assumption effectively precludes using updating algorithms based on approximate value iteration, Q-learning and least squares policy evaluation. We cannot eliminate the bias, but below we describe how to minimize it. We then describe Bayesian updating using lookup tables and parametric models.

17.7.1 Minimizing bias

We would very much like to have observations $\hat{v}^n(s)$ which we can view as an unbiased observation of $V(s)$. One way to do this is to build on the methods described in section 17.1.

To illustrate, assume that we have a policy π that determines the action a_t we take when in state S_t, generating a contribution $\hat{C}_t^n$. Assume we simulate this policy for T time periods using

$$\hat{v}^n(T) = \sum_{t=0}^{T} \gamma^t \hat{C}_t.$$

If we have a finite horizon problem and T is the end of our horizon, then we are done. If our problem has an infinite horizon, we can project the infinite horizon value of our policy by first approximating the one-period contribution using

$$\bar{c}_T^n = \frac{1}{T} \sum_{t=0}^{T} \hat{C}_t^n.$$

Now assume this estimates the average contribution per period starting at time $T+1$. Our infinite-horizon estimate would be

$$\hat{v}^n = \hat{v}_0(T) + \gamma^{T+1} \frac{1}{1-\gamma} \bar{c}_T^n.$$

Finally, we use $\hat{v}^n$ to update our value function approximation $\overline{V}^{n-1}$ to obtain $\overline{V}^n$.

We next illustrate the Bayesian updating formulas for lookup tables and parametric models.

17.7.2 Lookup tables with correlated beliefs

Previously when we have used lookup tables, if we update the value $\overline{V}^n(s)$ for some state s, we do not use this information to update the values of any other states. With our Bayesian model, we can do much more if we have access to a covariance function such as the one we illustrated in equation (17.51).

Assume that we have discrete states, and assume that we have a covariance function $Cov(s, s')$ in the form of a covariance matrix Σ where $Cov(s, s') = \Sigma(s, s')$. Let V^n be our vector of beliefs about the value $V(s)$ of being in each state (we use V^n to represent our Bayesian beliefs, so that $\overline{V}^n$ can represent our frequentist estimates). Also let Σ^n be the covariance matrix of our belief about the vector V. If $\hat{v}^n(S^n)$ is an (approximately) unbiased sample observation of $V(s)$, the Bayesian formula for updating V^n is given by

$$\overline{V}^{n+1}(s) = V^n(s) + \frac{\hat{v}^n(S^n) - V^n(s)}{\lambda(S^n) + \Sigma^n(S^n, S^n)} \Sigma^n(s, S^n).$$

This has to be computed for each s (or at least each s where $\Sigma^n(s, S^n) > 0$). We update the covariance matrix using

$$\Sigma^{n+1}(s, s') = \Sigma^n(s, s') - \frac{\Sigma^n(s, S^n)\Sigma^n(S^n, s')}{\lambda(S^n) + \Sigma^n(S^n, S^n)}.$$

17.7.3 Parametric models

For most applications, a parametric model (specifically, a linear model) is going to be much more practical. Our frequentist updating equations for our regression vector θ^n were given above as

$$\theta^n = \theta^{n-1} - \frac{1}{\gamma^n} B^{n-1} \phi^n \hat{\varepsilon}^n, \tag{17.52}$$

$$B^n = B^{n-1} - \frac{1}{\gamma^n}(B^{n-1}\phi^n(\phi^n)^T B^{n-1}), \tag{17.53}$$

$$\gamma^n = 1 + (\phi^n)^T B^{n-1} \phi^n, \tag{17.54}$$

where $\hat{\varepsilon}^n = \overline{V}(\theta^{n-1})(S^n) - \hat{v}^n$ is the difference between our current estimate $\overline{V}(\theta^{n-1})(S^n)$ of the value function at our observed state S^n and our most recent observation $\hat{v}^n$. The adaptation for a Bayesian model is quite minor. The matrix B^n represents

$$B^n = [(X^n)^T X^n]^{-1}.$$

It is possible to show that the covariance matrix Σ^θ (which is dimensioned by the number of basis functions) is given by

$$\Sigma^\theta = B^n \lambda.$$

In our Bayesian model, λ is the variance of the difference between our observation $\hat{v}^n$ and the true value function $v(S^n)$, where we assume λ is known. This variance may depend on the state that we have observed, in which case we would write it as $\lambda(s)$, but in practice, since we do not know the function $V(s)$, it is hard to believe that we would be able to specify $\lambda(s)$. We replace B^n with $\Sigma^{\theta,n}$ and rescale γ^n to create the following set of updating equations

$$\theta^n = \theta^{n-1} - \frac{1}{\gamma^n} \Sigma^{\theta,n-1} \phi^n \hat{\varepsilon}^n, \tag{17.55}$$

$$\Sigma^{\theta,n} = \Sigma^{\theta,n-1} - \frac{1}{\gamma^n}(\Sigma^{\theta,n-1}\phi^n(\phi^n)^T \Sigma^{\theta,n-1}), \tag{17.56}$$

$$\gamma^n = \lambda + (\phi^n)^T \Sigma^{\theta,n-1} \phi^n. \tag{17.57}$$

17.7.4 Creating the prior

Approximate dynamic programming has been approached from a Bayesian perspective in the research literature, but otherwise has apparently received very little attention. We suspect that while there exist many applications in stochastic search where it is valuable to use a prior distribution of belief, it is much harder to build a prior on a value function.

Lacking any specific structural knowledge of the value function, we anticipate that the easiest strategy will be to start with $V^0(s) = v^0$, which is a constant across all states. There are several strategies we might use to estimate v^0. We might sample a state S^i at random, and find the best contribution $\hat{C}^i = \max_a C(S^i, a)$. Repeat this n times and compute

$$\bar{c} = \frac{1}{n}\sum_{i=1}^{n}\hat{C}^i.$$

Finally, let $v^0 = \frac{1}{1-\gamma}\bar{c}$ if we have an infinite horizon problem. The hard part is that the variance λ has to capture the variance of the difference between v^0 and the true $V(s)$. This requires having some sense of the degree to which v^0 differs from $V(s)$. We recommend being very conservative, which is to say choose a variance λ such that $v^0+2\sqrt{\lambda}$ easily covers what $V(s)$ might be. Of course, this also requires some judgment about the likelihood of visiting different states.

17.8 LEARNING ALGORITHMS AND STEPSIZES

A useful exercise to understand the behavior of recursive least squares, LSTD and LSPE is to consider what happens when they are applied to a trivial dynamic program with a single state and a single action. Obviously, we are interested in the policy that chooses the single action. This dynamic program is equivalent to computing the sum

$$F = \mathbb{E}\sum_{i=0}^{\infty}\gamma^i\hat{C}^i, \tag{17.58}$$

where $\hat{C}^i$ is a random variable giving the ith contribution. If we let $\bar{c} = \mathbb{E}\hat{C}^i$, then clearly $F = \frac{1}{1-\gamma}\bar{c}$. But let's pretend that we do not know this, and we are using these various algorithms to compute the expectation.

We first used the single-state problem in section 17.4, but did not focus on the implications for stepsizes. Here, we use our ability to derive analytical solutions for the optimal value function for least squares temporal differences (LSTD), least squares policy evaluation (LSPE), and recursive least squares and temporal differences. These expressions allow us to understand the types of behaviors we would like to see in a stepsize formula.

In the remainder of this section, we start by assuming that the value function is approximated using a linear model

$$\overline{V}(s) = \sum_{f\in\mathcal{F}}\theta_f\phi_f(s).$$

However, we are going to then transition to a problem with a single state, and a single basis function $\phi(s) = 1$. We assume that $\hat{v}$ is a sampled estimate of the value of being in the single state.

17.8.1 Least squares temporal differences

In section 17.3 we showed that the LSTD method, when using a linear architecture, applied to infinite horizon problems required solving

$$\sum_{i=1}^{n} \phi_f(S^i)(\phi_f(S^i) - \gamma\phi_f(S^{i+1}))^T\theta = \sum_{i=1}^{n} \phi_f(S^i)\hat{C}^i,$$

for each $f \in \mathcal{F}$. Let θ^n be the optimal solution. Again, since we have only one basis function $\phi(s) = 1$ for our single state problem, this reduces to finding $v^n = \theta^n$

$$v^n = \frac{1}{1-\gamma}\left(\frac{1}{n}\sum_{i=1}^{n}\hat{C}^n\right). \tag{17.59}$$

Equation (17.59) shows that we are trying to estimate $\mathbb{E}\hat{C}$ using a simple average. If we let $\bar{C}^n$ be the average over n observations, we can write this recursively using

$$\bar{C}^n = \left(1 - \frac{1}{n}\right)\bar{C}^{n-1} + \frac{1}{n}\hat{C}^n.$$

For the single state (and single action) problem, the sequence $\hat{C}^n$ comes from a stationary sequence. In this case a simple average is the best possible estimator. In a dynamic programming setting with multiple states, and where we are trying to optimize over policies, v^n would depend on the state. Also, because the policy that determines the action we take when we are in a state is changing over the iterations, the observations $\hat{C}^n$, even when we fix a state, would be nonstationary. In this setting, simple averaging is no longer the best. Instead, it is better to use

$$\bar{C}^n = (1 - \alpha_{n-1})\bar{C}^{n-1} + \alpha_{n-1}\hat{C}^n, \tag{17.60}$$

and use one of the stepsizes described in section 6.1, 6.2 or 6.3. As a general rule, these stepsize rules do not decline as quickly as $1/n$.

17.8.2 Least squares policy evaluation

Least squares policy evaluation, which is developed using basis functions for infinite horizon applications, finds the regression vector θ by solving

$$\theta^n = \arg\min_\theta \sum_{i=1}^{n}\left(\sum_f \theta_f\phi_f(S^i) - \left(\hat{C}^i + \gamma\overline{V}^{n-1}(S^{i+1})\right)\right)^2.$$

When we have one state, the value of being in the single state is given by $v^n = \theta^n$ which we can write as

$$v^n = \arg\min_\theta \sum_{i=1}^{n}\left(\theta - \left(\hat{C}^i + \gamma v^{n-1}\right)\right)^2.$$

This problem can be solved in closed form, giving us

$$v^n = \left(\frac{1}{n}\sum_{i=1}^{n}\hat{C}^i\right) + \gamma v^{n-1}.$$

Similar to LSTD, LSPE works to estimate $\mathbb{E}\hat{C}$. For a problem with a single state and action (and therefore only one policy), the best estimate of $\mathbb{E}\hat{C}$ is a simple average. However, as we already argued with LSTD, if we have multiple states and are searching for the best policy, the observation $\hat{C}$ for a particular state will come from a nonstationary series. For such problems, we should again adopt the updating formula in (17.60) and use one of the stepsize rules described section 6.1, 6.2 or 6.3.

17.8.3 Recursive least squares

Using our linear model, we start by using the following standard least squares model to fit our approximation

$$\min_{\theta} \sum_{i=1}^{n} \left(\hat{v}^i - \left(\sum_{f \in \mathcal{F}} \theta_f \phi_f(S^i) \right) \right)^2 .$$

As we have already discussed in chapter 3, we can fit the parameter vector θ using least squares, which can be computed recursively using

$$\theta^n = \theta^{n-1} - \frac{1}{1 + (x^n)^T B^{n-1} x^n} B^{n-1} x^n (\overline{V}^{n-1}(S^n) - \hat{v}^n)$$

where $x^n = (\phi_1(S^n), \ldots, \phi_f(S^n), \ldots, \phi_F(S^n))$, and the matrix B^n is computed using

$$B^n = B^{n-1} - \frac{1}{1 + (x^n)^T B^{n-1} x^n} \left(B^{n-1} x^n (x^n)^T B^{n-1} \right).$$

For the special case of a single state, we use the fact that we have only one basis function $\phi(s) = 1$ and one parameter $\theta^n = \overline{V}^n(s) = v^n$. In this case, the matrix B^n is a scalar, and the updating equation for θ^n (now v^n), becomes

$$\begin{aligned} v^n &= v^{n-1} - \frac{B^{n-1}}{1 + B^{n-1}} (v^{n-1} - \hat{v}^n) \\ &= \left(1 - \frac{B^{n-1}}{1 + B^{n-1}} \right) v^{n-1} + \frac{B^{n-1}}{1 + B^{n-1}} \hat{v}^n. \end{aligned}$$

If $B^0 = 1$, $B^{n-1} = 1/n$, giving us

$$v^n = \left(1 - \frac{1}{n} \right) v^{n-1} + \frac{1}{n} \hat{v}^n. \tag{17.61}$$

Now imagine we are using approximate value iteration. In this case, $\hat{v}^n = \hat{C}^n + \gamma v^n$. Substituting this into equation (17.61) gives us

$$\begin{aligned} v^n &= \left(1 - \frac{1}{n} \right) v^{n-1} + \frac{1}{n} (\hat{C}^n + \gamma \hat{v}^n) \\ &= \left(1 - \frac{1}{n} (1 - \gamma) \right) v^{n-1} + \frac{1}{n} \hat{C}^n. \end{aligned} \tag{17.62}$$

Recursive least squares has the behavior of averaging the observations of $\hat{v}$. The problem is that $\hat{v}^n = \hat{C}^n + \gamma v^n$, since $\hat{v}^n$ is also trying to be a discounted accumulation of the costs.

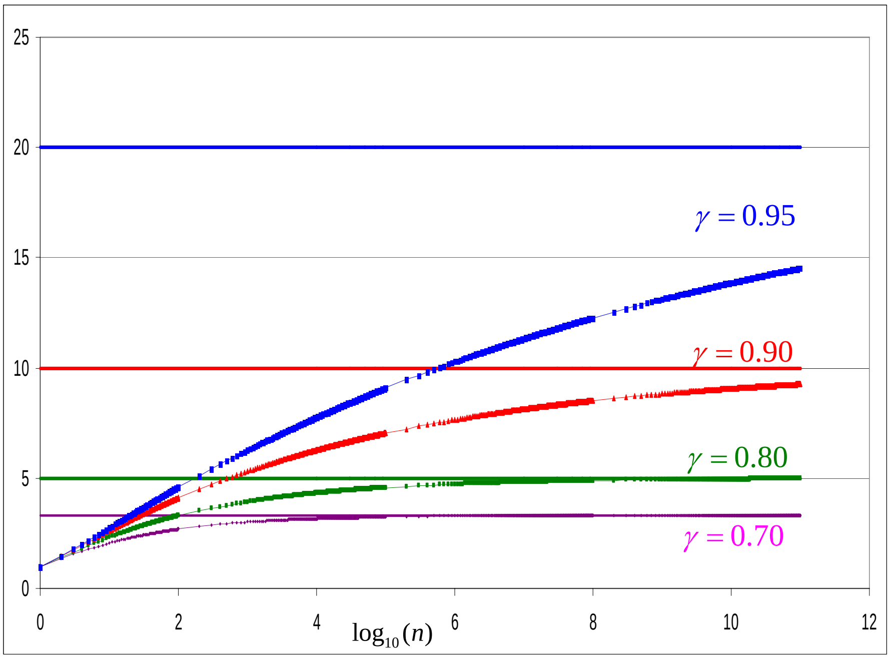


Figure 17.3 $\bar{v}^n$ plotted against $\log_{10}(n)$ when we use a $1/n$ stepsize rule for updating.

Assume that the contribution was deterministic, where $\hat{C} = c$. If we were doing classical approximate value iteration, we would write

$$v^n = c + \gamma v^{n-1}. \tag{17.63}$$

Comparing (17.62) and (17.63), we see that the one-period contribution carries a coefficient of $1/n$ in (17.62) and a coefficient of 1 in (17.62). We can view equation (17.62) as a steepest ascent update with a stepsize of $1/n$. If we change the stepsize to 1, we obtain (17.63).

17.8.4 Bounding $1/n$ convergence for approximate value iteration

It is well known that a $1/n$ stepsize will produce a provably convergent algorithm when used with approximate value iteration. Experimentalists know that the rate of convergence can be quite slow, but people new to the field can sometimes be found using this stepsize rule. In this section, we hope to present evidence that the $1/n$ stepsize should never be used with approximate value iteration or its variants.

Figure 17.3 is a plot of v^n computed using equation (17.62) as a function of $\log_{10}(n)$ for $\gamma = 0.7, 0.8, 0.9$, and 0.95, where we have set $\hat{C} = 1$. For $\gamma = 0.90$, we need 10^{10} iterations to get $\bar{v}^n = 9$, which means we are still 10 percent from the optimal. For $\gamma = 0.95$, we are not even close to converging after 100 billion iterations.

It is possible to derive compact bounds, $\nu^L(n)$ and $\nu^U(n)$ for $\bar{v}^n$ where

$$\nu^L(n) < v^n < \nu^U(n).$$

These are given by

$$\nu^L(n) \quad = \quad \frac{c}{1-\gamma}\left(1-\left(\frac{1}{1+n}\right)^{1-\gamma}\right), \tag{17.64}$$

$$\nu^U(n) \quad = \quad \frac{c}{1-\gamma}\left(1-\frac{1-\gamma}{\gamma n}-\frac{1}{\gamma n^{1-\gamma}}\left(\gamma^2+\gamma-1\right)\right). \tag{17.65}$$

Using the formula for the lower bound (which is fairly tight when n is large enough that v^n is close to v^*), we can derive the number of iterations to achieve a particular degree of accuracy. Let $\hat{C}=1$, which means that $v^*=1/(1-\gamma)$. For a value of $v<1/(1-\gamma)$, we would need at least $n(v)$ to achieve $\bar{v}^*=v$, where $n(v)$ is found (from (17.64)) to be

$$n(v) \geq [1-(1-\gamma)v]^{-1/(1-\gamma)}. \tag{17.66}$$

If $\gamma=0.9$, we would need $n(v)=10^{20}$ iterations to reach a value of $v=9.9$, which gives us a one percent error. On a 3-GHz chip, assuming we can perform one iteration per clock cycle (that is, 3×10^9 iterations per second), it would take 1,000 years to achieve this result.

17.8.5 Discussion

We can now see the challenge of choosing stepsizes for approximate value iteration, temporal-difference learning and Q-learning, compared to algorithms such as LSPE, LSTD and approximate policy iteration (the finite horizon version of LSPE). If we observe $\hat{C}$ with noise, and if the discount factor $\gamma=0$ (which means we are not trying to accumulate contributions over time), then a stepsize of $1/n$ is ideal. We are just averaging contributions to find the average value. As the noise in $\hat{C}$ diminishes, and as γ increases, we would like a stepsize that approaches 1. In general, we have to strike a balance between accumulating contributions over time (which is more important as γ increases) and averaging the observations of contributions (for which a stepsize of $1/n$ is ideal).

By contrast, LSPE, LSTD and approximate policy iteration are all trying to estimate the average contribution per period for each state. The values $\hat{C}(s,a)$ are nonstationary because the policy that chooses the action is changing, making the sequence $\hat{C}(s^n,a^n)$ nonstationary. But these algorithms are not trying to simultaneously accumulate contributions over time.

17.9 WHY DOES IT WORK*

17.10 BIBLIOGRAPHIC NOTES

Section 17.1 - This section reviews a number of classical methods for estimating the value of a policy drawn from the reinforcement learning community. The best overall reference for this is Sutton & Barto (1998). Least-squares temporal differencing is due to Bradtke & Barto (1996).

Section 17.2 - Tsitsiklis (1994) and Jaakkola et al. (1994*b*) were the first to make the connection between emerging algorithms in approximate dynamic programming (Q-learning, temporal difference learning) and the field of stochastic approximation theory (Robbins & Monro (1951), Blum (1954*a*), **?**).

Section 3.8 - L. & Soderstrom (1983) and Young (1984) provide nice treatments of recursive statistics. Precup et al. (2001) gives the first convergent algorithm for off-policy temporal-difference learning using basis functions by using an adjustment which based on the relative probabilities of choosing an action from the target and behavioral policies. Lagoudakis et al. (2002) and Bradtke & Barto (1996) present least squares methods in the context of reinforcement learning. Van Roy & Choi (2006) uses the Kalman filter to perform scaling for stochastic gradient updates, avoiding the scaling problems inherent in stochastic gradient updates such as equation (17.22). Nedic et al. (2003) describes the use of least squares equation with a linear (in the parameters) value function approximation using policy iteration and proves convergence for TD(λ) with general λ. Bertsekas et al. (2004) presents a scaled method for estimating linear value function approximations within a temporal differencing algorithm. Section **??** is based on Soderstrom et al. (1978).

Section 17.3 - The development of Bellman's equation using linear models is based on Tsitsiklis & Van Roy (1997), Lagoudakis & Parr (2003) and Bertsekas (2009). Tsitsiklis & Van Roy (1997) highlights the central role of the D-norm used in this section, which also plays a central role in the design of a simulation-based version of the algorithm.

Section 17.4 - The analysis of dynamic programs with a single state is based on Ryzhov et al. (2009).

Section 17.5 - Baird (1995) provides a nice example showing that approximate value iteration may diverge when using a linear architecture, even when the linear model may fit the true value function perfectly. Tsitsiklis & Van Roy (1997) establishes the importance of using Bellman errors weighted by the probability of being in a state. Van Roy & de Farias (2000) shows that there does not necessarily exist a fixed point to the projected form of Bellman's equation $\Phi\theta = \Pi\mathcal{M}\Phi\theta$ where $\mathcal{M}$ is the max operator. This paper also shows that a fixed point does exist for a projection operator Π_D defined with respect to the norm $\|\cdot\|_D$ which weights a state s with the probability d_s of being in this state. This results is first shown for a fixed policy, and then for a class of randomized policies. GTD2 and TDC are due to Sutton et al. (2009), with material from Sutton et al. (2008).

Section 17.6 - Our adaptation of least squares temporal differencing using kernel regression was presented in Ma & Powell (2010).

Section 17.7 - Dearden et al. (1998*b*) introduces the idea of using Bayesian updating for Q-learning. Dearden et al. (1998*a*) then considers model-based Bayesian learning. Our presentation is based on Ryzhov & Powell (2010) which introduces the idea of correlated beliefs.

Section 3.13.2 - The Sherman-Morrison updating formulas are given in a number of references, such as L. & Soderstrom (1983) and Golub & Loan (1996).

PROBLEMS

17.1 Consider a "Markov decision process" with a single state and single policy. Assume that we do not know the expected value of the contribution $\hat{C}$, but each time it is sampled, draw a sample realization from the uniform distribution between 0 and 20. Also assume a discount factor of $\gamma = 0.90$. Let $V = \sum_{t=0}^{\infty} \gamma^t \hat{C}_t$. The exercises below can be formed in a spreadsheet.

a) Estimate V using LSTD using 100 iterations.

b) Estimate V using LSPE using 100 iterations.

c) Estimate V using recursive least squares, executing the algorithm for 100 iterations.

d) Estimate V using temporal differencing (approximate value iteration) and a stepsize of $1/n^{\cdot 7}$.

e) Repeat (d) using a stepsize of $5/(5 + n - 1$.

d) Compare the rates of convergence of the different procedures.

17.2 Repeat the exercise above using a discount factor of 0.95.

17.3 We are going to walk through the derivation of the equations in section 17.5 assuming that there is a feature for each state, where $\phi_f(s) = 1$ if feature f corresponds to state s, and 0 otherwise. When asked for a sample of a vector or matrix, assume there are three states and three features. As above, let d_s^π be the probability of being in state s under policy π, and let D^π be the diagonal matrix consisting of the elements d_s^π.

a) What is the column vector ϕ if $s = 1$? What does $\phi\phi^T$ look like?

b) If d_s^π is the probability of being in state s under policy π, write out $\mathbb{E}[\phi\phi^T]$.

c) Write out the matrix Φ.

d) What is the projection matrix Π?

e) Write out equation (17.44) for $MSPBE(\theta)$.

CHAPTER 18

FORWARD ADP II: POLICY OPTIMIZATION

We are finally ready to tackle the problem of searching for good policies while simultaneously trying to produce good value function approximations. Our discussion is restricted to problems where the policy is based on the value function approximation, since chapter 12 has already addressed the strategy of direct policy search. The guiding principle in this chapter is that we can find good policies if we can find good value function approximations.

The statistical tools presented in chapter 3 focused on finding the best statistical fit within a particular approximation architecture. We actually did not address whether we had chosen a good architecture. This is particularly true of our linear models, where we used the tools of stochastic optimization and linear regression to ensure that we obtain the best fit given a model, without regard to whether it was a good model.

In this chapter, we return to the problem of trying to find the best policy, where we assume throughout that our policies are of the form

$$A^\pi(S) = \arg\max_{a \in \mathcal{A}} \big(C(S,a) + \gamma \mathbb{E}\overline{V}(S^M(S,a,W))\big),$$

if we have an infinite horizon problem with discrete actions. Or, we may consider finite-horizon problems with vector-valued decisions x_t, where a policy would look like

$$X^\pi_t(S_t) = \arg\max_{x_t \in \mathcal{X}_t} \big(C(S_t,x_t) + \gamma \mathbb{E}\overline{V}_t(S^M(S_t,x_t,W_{t+1}))\big).$$

The point is that the policy depends on some sort of value function approximation. When we write our generic optimization problem

$$\max_{\pi} \mathbb{E} \sum_{t=0}^{T} \gamma^t C(S_t, A_t^{\pi}(S_t)),$$

the maximization over policies can mean choosing an architecture for $\overline{V}_t(S_t)$, and choosing the parameters that control the architecture. For example, we might be choosing between a myopic policy, or perhaps a simple linear architecture with one basis function

$$\overline{V}(S) = \theta_0 + \theta_1 S, \tag{18.1}$$

or perhaps a linear architecture with two basis functions,

$$\overline{V}(S) = \theta_0 + \theta_1 S + \theta_2 S^2. \tag{18.2}$$

We might even use a nonlinear architecture such as

$$\overline{V}(S) = \frac{e^{\theta_0 + \theta_1 S}}{1 + e^{\theta_0 + \theta_1 S}}.$$

Optimizing over policies may consist of choosing a value function approximation such as (18.1) or (18.2), but then we still have to choose the best parameter vector within each class.

We begin our presentation with an overview of the basic algorithmic strategies that we cover in this chapter, all of which are based on using value function approximations that are intended to approximate the value of being in a state. The remainder of the chapter is organized around covering the following strategies:

Approximate value iteration - These are policies that iteratively update the value function approximation, and then immediately update the policy. We strive to find a value function approximation that estimates the value of being in each state while following a (near) optimal policy, but only in the limit. We intermingle the treatment of finite and infinite horizon problems. Variations include

- **Lookup table representations** - Here we introduce three major strategies that reflect the use of the pre-decision state, state-action pairs, and the post-decision state:
  - **AVI for pre-decision state** - Approximate value iteration using the classical pre-decision state variable.
  - **Q-learning** - Estimating the value of state-action pairs.
  - **AVI for the post-decision state** - Approximate value iteration where value function approximations are approximated around the post-decision state.
- **Parametric architectures** - We summarize some of the extensive literature which depends on linear models (basis functions), and touch on nonlinear models.

Approximate policy iteration - These are policies that attempt to explicitly approximate the value of a policy to some level of accuracy within an inner loop, within which the policy is held fixed.

- **API using lookup tables** - We use this setting to present the basic idea.

API using linear models - This is perhaps one of the most important areas of research in approximate dynamic programming.

API using nonparametric models - This is a relatively young area of research, and we summarize some recent results.

The linear programming method - The linear programming method, first introduced in chapter 14, can be adapted to exploit value function approximations.

18.1 OVERVIEW OF ALGORITHMIC STRATEGIES

The algorithmic strategies that we examine in this chapter are based on the principles of value iteration and policy iteration, first introduced in chapter 14. We continue to adapt our algorithms to finite and infinite horizons. Basic value iteration for finite horizon problems work by solving

$$V_t(S_t) = \max_{a_t} \big(C(S_t, a_t) + \gamma \mathbb{E}\{V_{t+1}(S_{t+1})|S_t\} \big). \tag{18.3}$$

Equation (18.3) works by stepping backward in time, where $V_t(S_t)$ is computed for each (presumably discrete) state S_t. This is classical "backward" dynamic programming which suffers from the well known curse of dimensionality, because we typically are unable to "loop over all the states."

Approximate dynamic programming approaches finite horizon problems by solving problems of the form

$$\hat{v}_t^n = \max_{a_t} \big(C(S_t^n, a_t) + \gamma \overline{V}_{t+1}^{n-1}(S^{M,a}(S_t^n, a_t)) \big). \tag{18.4}$$

Here, we have formed the value function approximation around the post-decision state. We execute the equations by stepping forward in time. If a_t^n is the action that optimizes (18.4), then we compute our next state using $S_{t+1}^n = S^M(S_t^n, a_t^n, W_{t+1}^n)$ where W_{t+1}^n is sampled from some distribution. The process runs until we reach the end of our horizon, at which point we return to the beginning of the horizon and repeat the process.

Classical value iteration for infinite horizon problems is centered on the basic iteration

$$V^n(S) = \max_a \big(C(S, a) + \gamma \mathbb{E}\{V^{n-1}(S')|S\} \big). \tag{18.5}$$

Again, equation (18.5) has to be executed for each state S. After each iteration, the new estimate V^n replaces the old estimate V^{n-1} on the right, after which n is incremented.

When we use approximate methods, we might observe an estimate of the value of being in a state using

$$\hat{v}^n = \max_a \big(C(S^n, a) + \gamma \overline{V}^{n-1}(S^{M,a}(S^n, a^n)) \big). \tag{18.6}$$

We then use the observed state-value pair $(S^n, \hat{v}^n)$ to update the value function approximation.

When we use approximate value iteration, $\hat{v}^n$ (or $\hat{v}_t^n$) cannot be viewed as a noisy but unbiased observation of the value of being in a state. These observations are calculated as a function of the value function $\overline{V}^{n-1}(s)$. While we hope the value function approximation converges to something, we generally cannot say anything about the function prior to convergence. This means that $\hat{v}^n$ does not have any particular property. We are simply guided by the basic value iteration update in equation (18.5) (or (18.3)), which suggests that if we repeat this step often enough, we may eventually learn the right value function for the right policy. Unfortunately, we have only limited guarantees that this is the case when we depend on approximations.

Approximate value iteration imbeds a policy approximate loop within an outer loop where policies are updated. Assume we fix our policy using

$$A^{\pi,n}(S) = \arg\max_{a\in\mathcal{A}} \big(C(S,a) + \gamma \overline{V}^{n-1}(S^{M,a}(S,a))\big), \tag{18.7}$$

Now perform the loop over $m = 1, \ldots, M$

$$\hat{v}^{n,m} = \max_{a\in\mathcal{A}} \big(C(S^{n,m},a) + \gamma \overline{V}^{n-1}(S^{M,a}(S^{n,m},a))\big)$$

where $S^{n,m+1} = S^M(S^n, a^n, W^{n+1})$. Note that the value function $\overline{V}^{n-1}(s)$ remains constant within this inner loop. After executing this loop, we take the series of observations $\hat{v}^{n,1}, \ldots, \hat{v}^{n,M}$ and use them to update $\overline{V}^{n-1}(s)$ to obtain $\overline{V}^n(s)$. Typically, $\overline{V}^n(s)$ does not depend on $\overline{V}^{n-1}(s)$, other than to influence the calculation of $\hat{v}^{n,m}$. If M is large enough, $\overline{V}^n(s)$ will represent an accurate approximation of the value of being in state s while following the policy in equation (18.7). In fact, it is specifically because of this ability to approximate a policy that approximate policy iteration is emerging as a powerful algorithmic strategy for approximate dynamic programming. However, the cost of using the inner policy evaluation loop can be significant, and for this reason approximate value iteration and its variants remain popular.

18.2 APPROXIMATE VALUE ITERATION AND Q-LEARNING USING LOOKUP TABLES

Arguably the most natural and elementary approach for approximate dynamic programming uses approximate value iteration. In this section we explore variations of approximate value iteration and a closely related algorithm widely known as Q-learning.

18.2.1 Value iteration using a pre-decision state variable

Classical value iteration (for a finite-horizon problem) estimates the value of being in a specific state S_t^n

$$\hat{v}_t^n = \max_{a_t} \big(C(S_t^n, a_t) + \gamma \mathbb{E}\{V_{t+1}(S_{t+1})|S_t^n\}\big), \tag{18.8}$$

where $S_{t+1} = S^M(S_t^n, x_t, W_{t+1}^n)$, and S_t^n is the state that we are in at time t, iteration n. We assume that we are following a sample path ω^n, where we compute $W_{t+1}^n = W_{t+1}(\omega^n)$. After computing $\hat{v}_t^n$, we update the value function using the standard equation

$$\overline{V}_t^n(S_t^n) = (1-\alpha_{n-1})\overline{V}^{n-1}(S_t^n) + \alpha_{n-1}\hat{v}_t^n. \tag{18.9}$$

If we sample states at random (rather than following the trajectory) and repeat equations (18.8) and (18.9), we will eventually converge to the correct value of being in each state.

```
Step 0. Initialization:
    Step 0a. Initialize $\overline{V}_t^0, \; t \in \mathcal{T}$.
    Step 0b. Set $n = 1$.
    Step 0c. Initialize $S^0$.
Step 1. Sample $\omega^n$.
    Step 2. Do for $t = 0, 1, \ldots, T$:
        Step 2a: Choose $\hat{\Omega}^n \subseteq \Omega$ and solve:
            $\hat{v}_t^n = \max_{a_t} \left( C_t(S_t^{n-1}, a_t) + \gamma \sum_{\hat{\omega} \in \hat{\Omega}^n} p^n(\hat{\omega}) \overline{V}_{t+1}^{n-1}(S^M(S_t^{n-1}, a_t, W_{t+1}(\hat{\omega}))) \right)$
            and let $a_t^n$ be the value of $a_t$ that solves the maximization problem.
        Step 2b: Compute:
            $S_{t+1}^n = S^M(S_t^n, a_t^n, W_{t+1}(\omega^n)).$
        Step 2c. Update the value function:
            $\overline{V}_t^n \leftarrow U^V(\overline{V}_t^{n-1}, S_t^n, \hat{v}_t^n)$
Step 3. Increment $n$. If $n \leq N$, go to Step 1.
Step 4. Return the value functions $(\overline{V}_t^n)_{t=1}^T$.
```

Figure 18.1 Approximate dynamic programming using a pre-decision state variable.

Note that we are assuming a finite-horizon model, and that we can compute the expectation exactly. When we can compute the expectation exactly, this is very close to classical value iteration, with the only exception that we are not looping over all the states at every iteration.

One reason to use the pre-decision state variable is that for some problems, computing the expectation is easy. For example, W_{t+1} might be a binomial random variable (did a customer arrive, did a component fail) which makes the expectation especially easy. If this is not the case, then we have to approximate the expectation. For example, we might use

$$\hat{v}_t^n = \max_{a_t} \left(C(S_t^n, a_t) + \gamma \sum_{\hat{\omega} \in \hat{\Omega}^n} p^n(\hat{\omega}) \overline{V}_{t+1}^{n-1}(S^M(S_t^n, a_t, W_{t+1}(\hat{\omega}))) \right). \tag{18.10}$$

Either way, using a lookup table representation we can update the value of being in state S_t^n using

$$\overline{V}_t^n(S_t^n) = (1 - \alpha_{n-1}) \overline{V}_t^{n-1}(S_t^n) + \alpha_{n-1} \hat{v}_t^n.$$

Keep in mind that if we can compute an expectation (or if we approximate it using a large sample $\hat{\Omega}$), then the stepsize should be much larger than when we are using a single sample realization (as we did with the post-decision formulation). An outline of the overall algorithm is given by figure 18.1.

At this point a reasonable question to ask is: Does this algorithm work? The answer is possibly, but not in general. Before we get an algorithm that will work (at least in theory), we need to deal with what is known as the exploration-exploitation problem.

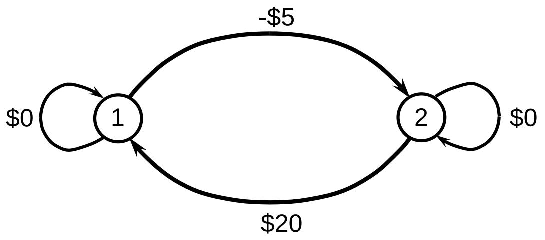


Figure 18.2 Two-state dynamic program, with transition contributions.

18.2.2 On-policy, off-policy and the exploration-exploitation problem

The algorithm in figure 18.1 uses a kind of default logic for determining the next state to visit. Specifically, we solve the optimization problem in equation (18.10) and from this, we not only determine $\hat{v}_t^n$ which we use to update the value of being in a state, we also determine an action a_t^n. Then, in Step 2b of the algorithm, we use this action to help determine the next state to visit using the transition function

$$S_{t+1}^n = S^M(S_t^n, a_t^n, W_{t+1}^n).$$

Using the action a_t^n, which is the action determined by the policy we are trying to optimize, means that we are using a concept known as *trajectory following* . The policy that determines the action we would like to take is known in the reinforcement learning community as the *target policy*. When we are optimizing policies, what we are doing is trying to improve the target policy. When we are approximating the value of being in a state while following a fixed policy, we are evaluating the target policy, which is sometimes referred to as the *learning policy*.

We can encounter serious problems if we use the target policy to determine the next state to visit. Consider the two-stage dynamic program illustrated in figure 18.2. Assume we start in state 1, and further assume that we initialize the value of being in each of the two states to $\overline{V}^0(1) = \overline{V}^0(2) = 0$. We see a negative contribution of -$5 to move from state 1 to 2, but a contribution of $0 to stay in state 1. We do not see the contribution of $20 to move from state 2 back to state 1, to it appears to be best to stay in state 1.

We need some way to force the system to visit state 2, so that we discover the contribution of $20. One way to do this is to adopt logic that forces the system to *explore* by choosing actions at random. For example, we may flip a coin and choose an action with probability ϵ, or choose the action a_t^n determined by the target policy with probability $1 - \epsilon$. This policy is known in the literature as *epsilon greedy*.

The policy that determines which action to use to determine the next state to visit, if it is different than the target policy, is known as the *behavior policy* or the *sampling policy*. The name “behavior policy” arises when we are modeling a real system such as a human playing a game or a factory assembling components. The behavior policy is, literally, the policy that describes how the system behaves. By contrast, if we are simply designing an algorithm, we feel that the term “sampling policy” more accurately describes the actual function being served by this policy. We also note that while it is common to implement a sampling policy through the choice of action, we may also simply choose a state at random.

If the target policy also determines the next state we visit, then we say that the algorithm is *on policy*. If the sampling policy is different than the target policy, then we say that the algorithm is *off policy*, which means that the policy that we use to determine the next state to visit does not follow the policy we are trying to optimize.

In the remainder of this chapter, we are going to make a distinction between on-policy and off-policy algorithms.

18.2.3 Q-learning

One of the earliest and most widely studied algorithms in the reinforcement learning literature is known as Q-learning. The name is derived simply from the notation used in the algorithm, and appears to have initiated the tradition of naming algorithms after the notation.

To motivate Q-learning, return for the moment to the classical way of making decisions using dynamic programming. Normally we would want to solve

$$a_t^n \quad = \quad \arg\max_{a_t \in \mathcal{A}_t^n} \Big\{ C_t(S_t^n, a_t) + \gamma \mathbb{E} \overline{V}_{t+1}^{n-1} \left(S_{t+1}(S_t^n, a_t, W_{t+1}) \right) \Big\}. \tag{18.11}$$

Solving equation (18.11) can be problematic for two different reasons. The first is that we may not be able to compute the expectation because it is computationally too complex (the second curse of dimensionality). The second is that we may simply not have the information we need to compute the expectation. This might happen if a) we do not know the probability distribution of the random information or b) we may not know the transition function. In either of these cases, we say that we do not "know the model" and we need to use a "model-free" formulation. When we can compute the expectation, which means we have the transition function and we know the probability distribution, then we are using what is known as a "model-based" formulation. Many authors equate "model-based" with knowing the one-step transition matrix, but this ignores the many problems where we know the transition function, we know the probability law for the exogenous information, but we simply cannot compute the transition function either because the state space is too large (or continuous), or the exogenous information is multidimensional.

Earlier, we circumvented this problem by approximating the expectation by using a subset of outcomes (see equation (18.10)), but this can be computationally clumsy for many problems. One thought is to solve the problem for a single sample realization

$$a_t^n \quad = \quad \arg\max_{a_t \in \mathcal{A}_t^n} \big(C_t(S_t^n, a_t) + \gamma \overline{V}_{t+1}^{n-1} (S_{t+1}(S_t^n, a_t, W_{t+1}(\omega^n))) \big). \tag{18.12}$$

The problem is that this means we are choosing a_t for a particular realization of the future information $W_{t+1}(\omega^n)$. If we use the same sample realization of $W_{t+1}(\omega^n)$ to make the decision that will actually happen (when we step forward in time), then this is what is known as cheating (peeking into the future), which can seriously distort the behavior of the system. If we use a single sample realization for $W_{t+1}(\omega)$ that is different than the one we use when we simulate forward, then this is simply unlikely to produce good results (imagine computing averages based on a single observation).

What if we instead choose the decision a_t^n first, then observe W_{t+1}^n (so we are not using this information when we choose our action) and then compute the cost? Let the resulting cost be computed using

$$\hat{q}_t^n(S_t, a_t) = C(S_t, a_t) + \gamma \max_{a_{t+1}} \overline{V}_{t+1}^{n-1}(S^M(S_t^n, a_t, W_{t+1}(\omega^n))). \tag{18.13}$$

```
Step 0. Initialization:
    Step 0a. Initialize an approximation for the value function $\bar{Q}_t^0(S_t, a_t)$ for all states $S_t$ and decisions
        $a_t \in \mathcal{A}_t, t = \{0, 1, \ldots, T\}$.
    Step 0b. Set $n = 1$.
    Step 0c. Initialize $S_0^1$.
Step 1. Choose a sample path $\omega^n$.
    Step 2. Do for $t = 0, 1, \ldots, T$:
        Step 2a: Determine the action using $\epsilon$-greedy. With probability $\epsilon$, choose an action $a^n$ at random
            from $\mathcal{A}$. With probability $1 - \epsilon$, choose $a^n$ using
                $a_t^n = \arg\max_{a_t \in \mathcal{A}_t} \bar{Q}_t^{n-1}(S_t^n, a_t).$
        Step 2b. Sample $W_{t+1}^n = W_{t+1}(\omega^n)$ and compute the next state $S_{t+1}^n = S^M(S_t^n, a_t^n, W_{t+1}^n)$.
        Step 2c. Compute:
                $\hat{q}_t^n = C(S_t^n, a_t^n) + \gamma \max_{a_{t+1} \in \mathcal{A}_{t+1}} \bar{Q}_{t+1}^{n-1}(S_{t+1}^n, a_{t+1}).$
        Step 2d. Update $\bar{Q}_t^{n-1}$ and $\overline{V}_t^{n-1}$ using:
                $\bar{Q}_t^n(S_t^n, a_t^n) = (1 - \alpha_{n-1})\bar{Q}_t^{n-1}(S_t^n, a_t^n) + \alpha_{n-1}\hat{q}_t^n$
Step 3. Increment $n$. If $n \leq N$ go to Step 1.
Step 4. Return the Q-factors $(\bar{Q}_t^n)_{t=1}^T$.
```

Figure 18.3 A Q-learning algorithm.

We could now smooth these values to obtain

$$\bar{Q}_t^n(S_t, a_t) = (1 - \alpha_{n-1})\bar{Q}_t^{n-1}(S_t^n, a_t^n) + \alpha_{n-1}\hat{q}_t^n(S_t, a_t).$$

Not surprisingly, we can compute the value of being in a state from the Q-factors using

$$\overline{V}_t^n(S_t) = \max_a \bar{Q}_t^n(S_t, a). \tag{18.14}$$

If we combine (18.14) and (18.13), we obtain

$$\hat{q}_t^n = C(S_t, a_t) + \gamma \max_{a_{t+1}} \bar{Q}^{n-1}(S_{t+1}, a_{t+1}),$$

where $S_{t+1} = S^M(S_t^n, a_t, W_{t+1}(\omega^n))$ is the next state resulting from action a_t and the sampled information $W_{t+1}(\omega^n)$.

The functions $Q_t(S_t, a_t)$ are known as *Q-factors* and they capture the value of being in a state and taking a particular action. We can now choose an action by solving

$$a_t^n = \arg\max_{a_t \in \mathcal{A}_t^n} \bar{Q}_t^{n-1}(S_t^n, a_t). \tag{18.15}$$

This strategy is known as Q-learning. The complete algorithm is summarized in figure 18.3.

A variation of Q-learning is known as "Sarsa." Imagine that we start in a state s and choose action a. After this, we observe a reward r and the next state s'. Finally, use some policy to choose the next action a'. This sequence forms "sarsa." A common strategy, given the initial state s, is to choose the action a and the action a' (given the state s') using an epsilon-greedy strategy. When a and a' are chosen using the same policy, the algorithm is referred to as on-policy. Alternatively, we may choose the initial action a at random, but then choose a' greedily, which is to say

$$a'_{t+1} = \arg\max_{a'} \bar{Q}^{n-1}_{t+1}(S', a').$$

If the policy used to choose a is different than the policy used to choose a', then we say that the algorithm is off-policy.

Many authors describe Q-learning as a technique for "model-free" dynamic programming, where we either do not know the transition function or the probability law of the exogenous information. The key idea is that we can choose an action using (18.15) without needing to directly approximate the future in any way. After choosing an action a^n (or a^n_t), we can then simply observe the next state we transition to, without an explicit model of how we got there.

18.2.4 Value iteration using a post-decision state variable

For the many applications that lend themselves to a compact post-decision state variable, it is possible to adapt approximate value iteration to value functions estimated around the post-decision state variable. At the heart of the algorithm we choose actions (and estimate the value of being in state S^n_t) using

$$\hat{v}^n_t \quad = \quad \arg\max_{a_t \in \mathcal{A}_t} \left(C(S^n_t, a_t) + \gamma \overline{V}^{n-1}_t (S^{M,a}(S^n_t, a_t)) \right).$$

The distinguishing feature when we use the post-decision state variable is that the maximization problem is now deterministic. The key step is how we update the value function approximation. Instead of using $\hat{v}^n_t$ to update a pre-decision value function approximation $\overline{V}^{n-1}(S^n_t)$, we use $\hat{v}^n_t$ to update a post-decision value function approximation around the *previous* post-decision state $S^{a,n}_{t-1}$. This is done using

$$\overline{V}^n_{t-1}(S^{a,n}_{t-1}) \quad = \quad (1 - \alpha_{n-1}) \overline{V}^{n-1}_{t-1}(S^{a,n}_{t-1}) + \alpha_{n-1} \hat{v}^n_t.$$

The post-decision state not only allows us to solve deterministic optimization problems, there are many applications where the post-decision state has either the same dimensionality as the pre-decision state, or, for some applications, a much lower dimensionality.

A complete summary of the algorithm is given in figure 18.4.

Q-learning shares certain similarities with dynamic programming using a post-decision value function. In particular, both require the solution of a deterministic optimization problem to make a decision. However, Q-learning accomplishes this goal by creating an artificial post-decision state given by the state/action pair (S, a). We then have to learn the value of being in (S, a), rather than the value of being in state S alone (which is already very hard for most problems).

If we compute the value function approximation $\overline{V}^n(S^a)$ around the post-decision state $S^a = S^{M,a}(S, a)$, we can create Q-factors directly from the contribution function and the post-decision value function using

$$\bar{Q}^n(S, a) = C(S, a) + \gamma \overline{V}^n_t (S^{M,a}(S, a)).$$

Viewed this way, approximate value iteration using value functions estimated around a post-decision state variable is equivalent to Q-learning. However, if the post-decision state is compact, then estimating $\overline{V}(S^a)$ is much easier than estimating $\bar{Q}(S, a)$.

```
Step 0. Initialization:
    Step 0a. Initialize an approximation for the value function $\overline{V}_t^0(S_t^a)$ for all post-decision states $S_t^a$,
        $t = \{0, 1, \dots, T\}$.
    Step 0b. Set $n = 1$.
    Step 0c. Initialize $S_0^{a,1}$.
Step 1. Choose a sample path $\omega^n$.
    Step 2. Do for $t = 0, 1, \dots, T$:
        Step 2a: Determine the action using $\epsilon$-greedy. With probability $\epsilon$, choose an action $a^n$ at random
            from $\mathcal{A}$. With probability $1 - \epsilon$, choose $a^n$ using
                $\hat{v}_t^n = \arg\max_{a_t \in \mathcal{A}_t} \left( C(S_t^n, a_t) + \gamma \overline{V}_t^{n-1}(S^{M,a}(S_t^n, a_t)) \right).$
            Let $a_t^n$ be the action that solves the maximization problem.
        Step 2b. Update $\overline{V}_{t-1}^{n-1}$ using:
                $\overline{V}_{t-1}^n(S_{t-1}^{a,n}) = (1 - \alpha_{n-1}) \overline{V}_{t-1}^{n-1}(S_{t-1}^{a,n}) + \alpha_{n-1} \hat{v}_t^n$
        Step 2c. Sample $W_{t+1}^n = W_{t+1}(\omega^n)$ and compute the next state $S_{t+1}^n = S^M(S_t^n, a_t^n, W_{t+1}^n)$.
Step 3. Increment $n$. If $n \leq N$ go to Step 1.
Step 4. Return the value functions $(\overline{V}_t^n)_{t=1}^T$.
```

Figure 18.4 Approximate value iteration for finite horizon problems using the post-decision state variable.

18.2.5 Value iteration using a backward pass

Classical approximate value iteration, which is equivalent to temporal difference learning with $\lambda = 0$ (also known as TD(0)), can be implemented using a pure forward pass, which enhances its simplicity. However, there are problems where it is useful to simulate decisions moving forward in time, and then updating value functions moving backward in time. This is also known as temporal difference learning with $\lambda = 1$, but we find "backward pass" to be more descriptive. The algorithm is depicted in figure 18.5.

In this algorithm, we step forward through time creating a trajectory of states, actions, and outcomes. We then step backwards through time, updating the value of being in a state using information from the same trajectory in the future. We are going to use this algorithm to also illustrate ADP for a time-dependent, finite horizon problem. In addition, we are going to illustrate a form of policy evaluation. Pay careful attention to how variables are indexed.

The idea of stepping backward through time to produce an estimate of the value of being in a state was first introduced in the control theory community under the name of *backpropagation through time* (BTT). The result of our backward pass is $\hat{v}_t^n$, which is the contribution from the sample path ω^n and a particular policy. Our policy is, quite literally, the set of decisions produced by the value function approximation $\overline{V}^{n-1}$. Unlike our forward-pass algorithm (where $\hat{v}_t^n$ depends on the approximation $\overline{V}_t^{n-1}(S_t^a)$), $\hat{v}_t^n$ is a valid, unbiased estimate of the value of being in state S_t^n at time t and following the policy produced by $\overline{V}^{n-1}$.

```
Step 0. Initialization:
    Step 0a. Initialize $\overline{V}^0_t, \;\; t \in \mathcal{T}$.
    Step 0b. Initialize $S^1_0$.
    Step 0c. Choose an initial policy $A^{\pi,0}$.
    Step 0d. Set $n = 1$.
Step 1. Repeat for $m = 1, 2, \ldots, M$:
    Step 1. Choose a sample path $\omega^m$.
    Step 2: Do for $t = 0, 1, 2, \ldots, T$:
        Step 2a: Find
            $a^{n,m}_t = A^{\pi,n-1}(S^n_t)$
        Step 2b: Update the state variable
            $S^{n,m}_{t+1} = S^M(S^{n,m}_t, a^{n,m}_t, W_{t+1}(\omega^n))$.
    Step 3: Set $\hat{v}^{n,m}_{T+1} = 0$ and do for $t = T, T-1, \ldots, 1$:
            $\hat{v}^{n,m}_t = C(S^{n,m}_t, a^{n,m}_t) + \gamma \hat{v}^{n,m}_{t+1}$.
Step 4: Compute the average value from starting in state $S^1_0$:
            $\bar{v}^n_0 = \frac{1}{M} \sum_{m=1}^{M} \hat{v}^{n,m}_0$.
Step 5. Update the value function approximation by using the average values:
            $\overline{V}^n_0 \leftarrow U^V(\overline{V}^{n-1}_0, S^{a,n}_0, \bar{v}^n_t)$.
Step 6. Update the policy
            $A^{\pi,n}(S) = \arg\max_{a \in \mathcal{A}} \left( C(S^n_0, a) + \gamma \overline{V}^n_0(S^{M,a}(S^n_0, a)) \right)$
Step 6. Increment $n$. If $n \leq N$ go to Step 1.
Step 7. Return the value functions $(\overline{V}^n_t)^T_{t=1}$.
```

Figure 18.5 Double-pass version of the approximate dynamic programming algorithm for a finite horizon problem.

We introduce an inner loop so that rather than updating the value function approximation with a single $\hat{v}^n_0$, we average across a set of samples to create a more stable estimate, $\bar{v}^n_0$.

These two strategies are easily illustrated using our simple asset selling problem. For this illustration, we are going to slightly simplify the model we provided earlier, where we assumed that the change in price, $\hat{p}_t$, was the exogenous information. If we use this model, we have to retain the price p_t in our state variable (even the post-decision state variable). For our illustration, we are going to assume that the exogenous information is the price itself, so that $p_t = \hat{p}_t$. We further assume that $\hat{p}_t$ is independent of all previous prices (a pretty strong assumption). For this model, the pre-decision state is $S_t = (R_t, p_t)$ while the post-decision state variable is simply $S^a_t = R^a_t = R_t - a_t$ which indicates whether we are holding the asset or not. Further, $S_{t+1} = S^a_t$ since the resource transition function is deterministic.

		$t=0$	$t=1$				$t=2$				$t=3$			
Iteration	α_{n-1}	$\bar{v}_0$	$\hat{v}_1$	$\hat{p}_1$	a_1	$\bar{v}_1$	$\hat{v}_2$	$\hat{p}_2$	a_2	$\bar{v}_2$	$\hat{v}_3$	$\hat{p}_3$	a_3	$\bar{v}_3$
0		0				0				0				0
1	1	30	30	30	1	34	34	34	1	31	31	31	1	0
2	0.50	31	32	24	0	31.5	29	21	0	29.5	30	30	1	0
3	0.3	32.3	35	35	1	30.2	27.5	24	0	30.7	33	33	1	0

Table 18.1 Illustration of a single-pass algorithm

With this model, a single-pass algorithm (approximate value iteration) is performed by stepping forward through time, $t = 1, 2, \ldots, T$. At time t, we first sample $\hat{p}_t$ and we find

$$\hat{v}_t^n = \max_{a_t \in \{0,1\}} \big(\hat{p}_t^n a_t + (1 - a_t)(-c_t + \bar{v}_t^{n-1})\big). \tag{18.16}$$

Assume that the holding cost $c_t = 2$ for all time periods.

Table 18.1 illustrates three iterations of a single-pass algorithm for a three-period problem. We initialize $\bar{v}_t^0 = 0$ for $t = 0, 1, 2, 3$. Our first decision is a_1 after we see $\hat{p}_1$. The first column shows the iteration counter, while the second shows the stepsize $\alpha_{n-1} = 1/n$. For the first iteration, we always choose to sell because $\bar{v}_t^0 = 0$, which means that $\hat{v}_t^1 = \hat{p}_t^1$. Since our stepsize is 1.0, this produces $\bar{v}_{t-1}^1 = \hat{p}_t^1$ for each time period.

In the second iteration, our first decision problem is

$$\begin{aligned} \hat{v}_1^2 &= \max\{\hat{p}_1^2, -c_1 + \bar{v}_1^1\} \\ &= \max\{24, -2 + 34\} \\ &= 32, \end{aligned}$$

which means $a_1^2 = 0$ (since we are holding). We then use $\hat{v}_1^2$ to update $\bar{v}_0^2$ using

$$\begin{aligned} \bar{v}_0^2 &= (1 - \alpha_1)\bar{v}_0^1 + \alpha_1 \hat{v}_1^1 \\ &= (0.5)30.0 + (0.5)32.0 \\ &= 31.0 \end{aligned}$$

Repeating this logic, we hold again for $t = 2$ but we always sell at $t = 3$ since this is the last time period. In the third pass, we again sell in the first time period, but hold for the second time period.

It is important to realize that this problem is quite simple, and we do not have to deal with exploration issues. If we sell, we are no longer holding the asset and the forward pass should stop (more precisely, we should continue to simulate the process given that we have sold the asset). Instead, even if we sell the asset, we step forward in time and continue to evaluate the state that we are holding the asset (the value of the state where we are not holding the asset is, of course, zero). Normally, we evaluate only the states that we transition to (see step 2b), but for this problem, we are actually visiting all the states (since there is, in fact, only one state that we really need to evaluate).

Now consider a double-pass algorithm. Table 18.2 illustrates the forward pass, followed by the backward pass, where for simplicity we are going to use only a single inner iteration ($M = 1$). Each line of the table only shows the numbers determined during the forward or

		$t = 0$	$t = 1$				$t = 2$				$t = 3$			
Iteration	Pass	$\bar{v}_0$	$\hat{v}_1$	$\hat{p}_1$	a_1	$\bar{v}_1$	$\hat{v}_2$	$\hat{p}_2$	a_2	$\bar{v}_2$	$\hat{v}_3$	$\hat{p}_3$	a_3	$\bar{v}_3$
0		0				0				0				0
1	Forward	→	→	30	1	→	→	34	1	→	→	31	1	
1	Back	30	30	←	←	34	34	←	←	31	31	←	←	0
2	Forward	→	→	24	0	→	→	21	0	→	→	27	1	
2	Back	26.5	23	←	←	29.5	25	←	←	29	27	←	←	0

Table 18.2 Illustration of a double-pass algorithm

backward pass. In the first pass, we always sell (since the value of the future is zero), which means that at each time period the value of holding the asset is the price in that period.

In the second pass, it is optimal to hold for two periods until we sell in the last period. The value $\hat{v}_t^2$ for each time period is the contribution of the rest of the trajectory which, in this case, is the price we receive in the last time period. So, since $a_1 = a_2 = 0$ followed by $a_3 = 1$, the value of holding the asset at time 3 is the \$27 price we receive for selling in that time period. The value of holding the asset at time $t = 2$ is the holding cost of -2 plus $\hat{v}_3^2$, giving $\hat{v}_2^2 = -2 + \hat{v}_3^2 = -2 + 27 = 25$. Similarly, holding the asset at time 1 means $\hat{v}_1^2 = -2 + \hat{v}_2^2 = -2 + 25 = 23$. The smoothing of $\hat{v}_t^n$ with $\bar{v}_{t-1}^{n-1}$ to produce $\bar{v}_{t-1}^n$ is the same as for the single pass algorithm.

The value of implementing the double-pass algorithm depends on the problem. For example, imagine that our asset is an expensive piece of replacement equipment for a jet aircraft. We hold the part in inventory until it is needed, which could literally be years for certain parts. This means there could be hundreds of time periods (if each time period is a day) where we are holding the part. Estimating the value of the part now (which would determine whether we order the part to hold in inventory) using a single-pass algorithm could produce extremely slow convergence. A double-pass algorithm would work dramatically better. But if the part is used frequently, staying in inventory for only a few days, then the single-pass algorithm will work fine.

18.3 STATISTICAL BIAS IN THE MAX OPERATOR

A subtle type of bias arises when we are optimizing because we are taking the maximum over a set of random variables. In algorithms such as Q-learning or approximate value iteration, we are computing $\hat{q}_t^n$ by choosing the best of a set of decisions which depend on $\bar{Q}^{n-1}(S, a)$. The problem is that the estimates $\bar{Q}^{n-1}(S, a)$ are random variables. In the best of circumstances, assume that $\bar{Q}^{n-1}(S, a)$ is an unbiased estimate of the true value $V_t(S^a)$ of being in (post-decision) state S^a. Because it is still a statistical estimate with some degree of variation, some of the estimates will be too high while others will be too low. If a particular action takes us to a state where the estimate just happens to be too high (due to statistical variation), then we are more likely to choose this as the best action and use it to compute $\hat{q}^n$.

To illustrate, assume we have to choose an action $a \in \mathcal{A}$, where $C(S, a)$ is the contribution earned by using decision a (given that we are in state S) which then takes us to state $S^a(S, a)$ where we receive an estimated value $\overline{V}(S^a(S, a))$. Normally, we would update the value of being in state S by computing

$$\hat{v}^n = \max_{a \in \mathcal{A}} \big(C(S,a) + \overline{V}^{n-1}(S^a(S,a)) \big).$$

We would then update the value of being in state S using our standard update formula

$$\overline{V}^n(S) = (1-\alpha_{n-1})\overline{V}^{n-1}(S) + \alpha_{n-1}\hat{v}^n.$$

Since $\overline{V}^{n-1}(S^a(S,a))$ is a random variable, sometimes it will overestimate the true value of being in state $S^a(S,a)$ while other times it will underestimate the true value. Of course, we are more likely to choose an action that takes us to a state where we have overestimated the value.

We can quantify the error due to statistical bias as follows. Fix the iteration counter n (so that we can ignore it), and let

$$U_a = C(S,a) + \overline{V}(S^a(S,a))$$

be the estimated value of using action a. The statistical error, which we represent as β, is given by

$$\beta = \mathbb{E}\{\max_{a \in \mathcal{A}} U_a\} - \max_{a \in \mathcal{A}} \mathbb{E} U_a. \tag{18.17}$$

The first term on the right-hand side of (18.17) is the expected value of $\overline{V}(S)$, which is computed based on the best observed value. The second term is the correct answer (which we can only find if we know the true mean). We can get an estimate of the difference by using a strategy known as the "plug-in principle." We assume that $\mathbb{E}U_a = \overline{V}(S^a(S,a))$, which means that we assume that the estimates $\overline{V}(S^a(S,a))$ are correct, and then try to estimate $\mathbb{E}\{\max_{a \in \mathcal{A}} U_a\}$. Thus, computing the second term in (18.17) is easy.

The challenge is computing $\mathbb{E}\{\max_{a \in \mathcal{A}} U_a\}$. We assume that while we have been computing $\overline{V}(S^a(S,a))$, we have also been computing $\bar{\sigma}^2(a) = \mathit{Var}(U_a) = \mathit{Var}\big(\overline{V}(S^a(S,a))\big)$. Using the plug-in principle, we are going to assume that the estimates $\bar{\sigma}^2(a)$ represent the true variances of the value function approximations. Computing $\mathbb{E}\{\max_{a \in \mathcal{A}} U_a\}$ for more than a few decisions is computationally intractable, but we can use a technique called the Clark approximation to provide an estimate. This strategy finds the exact mean and variance of the maximum of two normally distributed random variables, and then assumes that this maximum is also normally distributed. Assume the decisions can be ordered so that $\mathcal{A} = \{1, 2, \ldots, |\mathcal{A}|\}$. Now let

$$\bar{U}_2 = \max\{U_1, U_2\}.$$

We can compute the mean and variance of $\bar{U}_2$ as follows. First, we temporarily define α using

$$\alpha^2 = \sigma_1^2 + \sigma_2^2 - 2\sigma_1\sigma_2\rho_{12}$$

where $\sigma_1^2 = \mathit{Var}(U_1)$, $\sigma_2^2 = \mathit{Var}(U_2)$, and ρ_{12} is the correlation coefficient between U_1 and U_2 (we allow the random variables to be correlated, but shortly we are going to approximate them as being independent). Next find

$$z = \frac{\mu_1 - \mu_2}{\alpha}.$$

where $\mu_1 = \mathbb{E}U_1$ and $\mu_2 = \mathbb{E}U_2$. Now let $\Phi(z)$ be the cumulative standard normal distribution (that is, $\Phi(z) = \mathbb{P}[Z \leq z]$ where Z is normally distributed with mean 0 and variance 1), and let $\phi(z)$ be the standard normal density function. If we assume that U_1 and U_2 are normally distributed (a reasonable assumption when they represent sample estimates of the value of being in a state), then it is a straightforward exercise to show that

$$\begin{aligned} \mathbb{E}\bar{U}_2 &= \mu_1\Phi(z) + \mu_2\Phi(-z) + \alpha\phi(z) && (18.18) \\ \mathit{Var}(\bar{U}_2) &= \left[(\mu_1^2 + \sigma_1^2)\Phi(z) + (\mu_1^2 + \sigma_2^2)\Phi(-z) + (\mu_1 + \mu_2)\alpha\phi(z)\right] \\ &\quad -(\mathbb{E}\bar{U}_2)^2. && (18.19) \end{aligned}$$

Now assume that we have a third random variable, U_3, where we wish to find $\mathbb{E}\max\{U_1, U_2, U_3\}$. The Clark approximation solves this by using

$$\begin{aligned} \bar{U}_3 &= \mathbb{E}\max\{U_1, U_2, U_3\} \\ &\approx \mathbb{E}\max\{U_3, \bar{U}_2\}, \end{aligned}$$

where we assume that $\bar{U}_2$ is normally distributed with mean given by (18.18) and variance given by (18.19). For our setting, it is unlikely that we would be able to estimate the correlation coefficient ρ_{12} (or ρ_{23}), so we are going to assume that the random estimates are independent. This idea can be repeated for large numbers of decisions by using

$$\begin{aligned} \bar{U}_a &= \mathbb{E}\max\{U_1, U_2, \ldots, U_a\} \\ &\approx \mathbb{E}\max\{U_a, \bar{U}_{a-1}\}. \end{aligned}$$

We can apply this repeatedly until we find the mean of $\bar{U}_{|\mathcal{A}|}$, which is an approximation of $\mathbb{E}\{\max_{a\in\mathcal{A}} U_a\}$. This, in turn, allows us to compute an estimate of the statistical bias β given by equation (18.17).

Figure 18.6 plots $\beta = \mathbb{E}\max_a U_a - \max_a \mathbb{E}U_a$ as it is being computed for 100 decisions, averaged over 30 sample realizations. The standard deviation of each U_a was fixed at $\sigma = 20$. The plot shows that the error increases steadily until the set $\mathcal{A}$ reaches about 20 or 25 decisions, after which it grows much more slowly. Of course, in an approximate dynamic programming application, each U_a would have its own standard deviation which would tend to decrease as we sample a decision repeatedly (a behavior that the approximation above captures nicely).

This brief analysis suggests that the statistical bias in the max operator can be significant. However, it is highly data dependent. If there is a single dominant decision, then the error will be negligible. The problem only arises when there are many (as in 10 or more) decisions that are competitive, and where the standard deviation of the estimates is not small relative to the differences between the means. Unfortunately, this is likely to be the case in most large-scale applications (if a single decision is dominant, then it suggests that the solution is probably obvious).

The relative magnitudes of value iteration bias over statistical bias will depend on the nature of the problem. If we are using a pure forward pass (TD(0)), and if the value of being in a state at time t reflects rewards earned over many periods into the future, then the value iteration bias can be substantial (especially if the stepsize is too small). Value iteration bias has long been recognized in the dynamic programming community. By contrast, statistical bias appears to have received almost no attention, and as a result we are not aware of any research addressing this problem. We suspect that statistical bias is likely to inflate value function approximations fairly uniformly, which means that the impact on the policy may be quite small. However, if the goal is to obtain the value function itself (for example, to estimate the value of an asset or a contract), then the bias can distort the results.

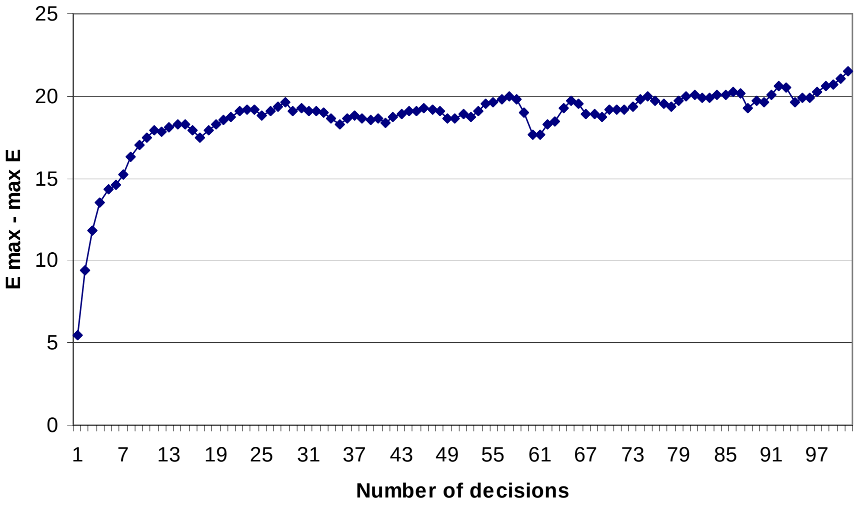


Figure 18.6 $\mathbb{E}\max_a U_a - \max_a \mathbb{E}U_a$ for 100 decisions, averaged over 30 sample realizations. The standard deviation of all sample realizations was 20.

```
Step 0. Initialization:
    Step 0a. Initialize $\overline{V}^0$.
    Step 0b. Initialize $S^1$.
    Step 0c. Set $n = 1$.
    Step 1. Solve
        $$\hat{v}^n = \max_{a \in \mathcal{A}^n} \left( C(S^n, a) + \gamma \sum_f \theta_f^{n-1} \phi_f(S^{M,a}(S^n, a)) \right) \qquad (18.20)$$
        and let $a^n$ be the value of $a$ that solves (18.20).
    Step 2. Update the value function recursively using equations (3.57) -(3.61) from chapter 17 to obtain $\theta^n$.
    Step 3. Choose a sample $W^{n+1} = W(\omega^{n+1})$ and determine the next state using some policy such as
        $$S^n = S^M(S^n, a^n, W^{n+1}).$$
Step 3. Increment $n$. If $n \leq N$ go to Step 1.
Step 4. Return the value functions $\overline{V}^N$.
```

Figure 18.7 Approximate value iteration using a linear model.

18.4 APPROXIMATE VALUE ITERATION USING LINEAR MODELS

Approximate value iteration, Q-learning and temporal difference learning (with $\lambda = 0$) are clearly the simplest methods for updating an estimate of the value of being in a state. Linear models are the simplest methods for approximating a value function. Not surprisingly, then, there has been considerable interest in putting these two strategies together.

Figure 18.7 depicts a basic adaptation of linear models updated using recursive least squares in an approximate value iteration. However, not only are there no convergence proofs for this algorithm, there are examples that show that it may not diverge, even for problems where the linear approximation has the potential for identifying the correct value function. As a result, approximate value iteration using a linear model is not a sound algorithmic strategy.

Despite the potentially poor performance of this algorithm, it remains a popular strategy because it is so easy to implement. In addition, while it may not work, on the other hand it *might* work quite well! The important point here is that it is a strategy that may be worth trying, but caution needs to be used. Given these observations, this section is aimed at providing some guidance to improve the chances that the algorithm will work.

The most important step whenever a linear model is used, regardless of the setting, is to choose the basis functions carefully so that the linear model has a chance of representing the true value function accurately. The biggest strength of a linear model is also its biggest weakness. A large error can distort the update of θ^n which then impacts the accuracy of the entire approximation. Since the value function approximation determines the policy (see Step 1), a poor approximation leads to poor policies, which then distorts the observations $\hat{v}^n$. This can be a vicious circle from which the algorithm may never recover.

A second step is in the specific choice of recursive least squares updating. Figure 18.7 refers to the classic recursive least squares updating formulas in equations (3.57)-(3.61). However, buried in these formulas is the implicit use of a stepsize rule of $1/n$. We show in chapter 6 that a stepsize $1/n$ is particularly bad for approximate value iteration (as well as Q-learning and TD(0) learning). While this stepsize can work well (indeed, it is optimal) for stationary data, it is very poorly suited for the backward learning that arises in approximate value iteration. Fortunately, the problem is easily fixed if we replace the updating equations for B^n and γ, which are given as

$$\begin{aligned} B^n &= B^{n-1} - \frac{1}{\gamma^n}(B^{n-1}\phi^n(\phi^n)^T B^{n-1}), \\ \gamma^n &= 1 + (\phi^n)^T B^{n-1}\phi^n, \end{aligned}$$

in equations (3.60) and (3.61) with

$$\begin{aligned} B^n &= \frac{1}{\lambda}\left(B^{n-1} - \frac{1}{\gamma^n}(B^{n-1}\phi^n(\phi^n)^T B^{n-1})\right), \\ \gamma^n &= \lambda + (\phi^n)^T B^{n-1}\phi^n, \end{aligned}$$

in equations (3.63) and (3.64). Here, λ discounts older errors. $\lambda = 1$ produces the original recursive formulas. When used with approximate value iteration, it is important to use $\lambda < 1$. In section 3.8.2, we argue that if you choose a stepsize rule for α_n such as $\alpha_n = a/(a+n-1)$, you should set λ_n at iteration n using

$$\lambda_n = \alpha_{n-1}\left(\frac{1-\alpha_n}{\alpha_n}\right).$$

The last issue that needs special care is the rule for determining the next state to visit. In step 3, we use $S^n = S^M(S^n, a^n, W(\omega^n))$ which we are using what we think is our best action to determine the next state. This strategy is known in the reinforcement learning community as an "on-policy" algorithm, which means that the action that is used to update the value of being in a state is also used to determine the next state to visit. It is very easy to create examples where this strategy will not work, and it is further complicated if there are structural problems with our linear model.

The issue of choosing the next state to visit is known as the exploration vs. exploitation problem. We first encountered this in chapter 7. How it should be solved is problem dependent. If you have a small action space, a standard strategy is to use ϵ-greedy, which we first mentioned briefly in chapter 11. In this strategy, we choose an action at random with probability ϵ, or use our optimal action a^n from step 1 with probability $1 - \epsilon$. If the state variable is continuous, we might simply add a noise term to force an exploration to nearby states. See chapter 7 for a thorough discussion of the issues and other strategies that can be used.

Approximate value iteration using a linear architecture has to be used with extreme care. Provable convergence results are rare, and there are examples of divergence. This does not mean the strategy should not be used, but its performance will be very problem dependent. It is particularly valuable to design some sort of benchmark (such as using lookup tables for a simplified version of the problem) against which your results can be compared. If your algorithm appears successful (for example, it outperforms a human, or provides high value), then you have a success. But if it performs poorly, it is impossible to know if this is the best you can do, or if your algorithm simply is not finding a good policy. It is precisely because of this lack of convergence guarantees that the research community has devoted increasing attention to approximate policy iteration.

18.4.1 Illustrations using regression models

There are many problems where we can exploit structure in the state variable, allowing us to propose functions characterized by a small number of parameters which have to be estimated statistically. Section 3.6.3 represented one version where we had a parameter for each (possibly aggregated) state. The only structure we assumed was implicit in the ability to specify a series of one or more aggregation functions.

The remainder of this section illustrates the use of regression models in specific applications. The examples use a specific method for estimating the parameter vector θ that will typically prove to be somewhat clumsy in practice.

Pricing an American option

Consider the problem of determining the value of an American-style put option which gives us the right to sell an asset (or contract) at a specified price at any of a set of discrete time periods. For example, we might be able to exercise the option on the last day of the month over the next 12 months.

Assume we have an option that allows us to sell an asset at \$1.20 at any of four time periods. We assume a discount factor of 0.95 to capture the time value of money. If we wait until time period 4, we must exercise the option, receiving zero if the price is over \$1.20. At intermediate periods, however, we may choose to hold the option even if the price is below \$1.20 (of course, exercising it if the price is above \$1.20 does not make sense). Our problem is to determine whether to hold or exercise the option at the intermediate points.

From history, we have found 10 samples of price trajectories which are shown in table 18.3.

If we wait until time period 4, our payoff is shown in table 18.4, which is zero if the price is above 1.20, and $1.20 - p_4$ for prices below \$1.20.

At time $t = 3$, we have access to the price history (p_1, p_2, p_3). Since we may not be able to assume that the prices are independent or even Markovian (where p_3 depends only

Stock prices				
	Time period			
Outcome	1	2	3	4
1	1.21	1.08	1.17	1.15
2	1.09	1.12	1.17	1.13
3	1.15	1.08	1.22	1.35
4	1.17	1.12	1.18	1.15
5	1.08	1.15	1.10	1.27
6	1.12	1.22	1.23	1.17
7	1.16	1.14	1.13	1.19
8	1.22	1.18	1.21	1.28
9	1.08	1.11	1.09	1.10
10	1.15	1.14	1.18	1.22

Table 18.3 Ten sample realizations of prices over four time periods

on p_2), the entire price history represents our state variable, along with an indicator that tells us if we are still holding the asset. We wish to predict the value of holding the option at time $t = 4$. Let $V_4(S_4)$ be the value of the option if we are holding it at time 4, given the state (which includes the price p_4) at time 4. Now let the conditional expectation at time 3 be

$$\overline{V}_3(S_3) \quad = \quad \mathbb{E}\{V_4(S_4)|S_3\}.$$

Our goal is to approximate $\overline{V}_3(S_3)$ using information we know at time 3. We propose a linear regression of the form

$$Y = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + \theta_3 X_3,$$

where

$$\begin{aligned} Y &= V_4, \\ X_1 &= p_2, \\ X_2 &= p_3, \\ X_3 &= (p_3)^2. \end{aligned}$$

The variables X_1, X_2 and X_3 are our basis functions. Keep in mind that it is important that our explanatory variables X_i must be a function of information we have at time $t = 3$, whereas we are trying to predict what will happen at time $t = 4$ (the payoff). We would then set up the data matrix given in table 18.5.

We may now run a regression on this data to determine the parameters $(\theta_i)_{i=0}^3$. It makes sense to consider only the paths which produce a positive value in the fourth time period,

Option value at $t = 4$				
	Time period			
Outcome	1	2	3	4
1	-	-	-	0.05
2	-	-	-	0.07
3	-	-	-	0.00
4	-	-	-	0.05
5	-	-	-	0.00
6	-	-	-	0.03
7	-	-	-	0.01
8	-	-	-	0.00
9	-	-	-	0.10
10	-	-	-	0.00

Table 18.4 The payout at time 4 if we are still holding the option

since these represent the sample paths where we are most likely to still be holding the asset at the end. The linear regression is only an approximation, and it is best to fit the approximation in the region of prices which are the most interesting (we could use the same reasoning to include some "near misses"). We only use the value function to estimate the value of holding the asset, so it is this part of the function we wish to estimate. For our illustration, however, we use all 10 observations, which produces the equation

$$\overline{V}_3 \approx 0.0056 - 0.1234p_2 + 0.6011p_3 - 0.3903(p_3)^2.$$

$\overline{V}_3$ is an approximation of the expected value of the price we would receive if we hold the option until time period 4. We can now use this approximation to help us decide what to do at time $t = 3$. Table 18.6 compares the value of exercising the option at time 3 against holding the option until time 4, computed as $\gamma\overline{V}_3(S_3)$. Taking the larger of the two payouts, we find, for example, that we would hold the option given samples 1-4, 6, 8, and 10, but would sell given samples 5, 7, and 9.

We can repeat the exercise to estimate $\overline{V}_2(S_t)$. This time, our dependent variable "Y" can be calculated two different ways. The simplest is to take the larger of the two columns from table 18.6 (marked in bold). So, for sample path 1, we would have $Y_1 = \max\{.03, 0.03947\} = 0.03947$. This means that our observed value is actually based on our approximate value function $\overline{V}_3(S_3)$.

An alternative way of computing the observed value of holding the option in time 3 is to use the approximate value function to determine the decision, but then use the actual price we receive when we eventually exercise the option. Using this method, we receive 0.05 for the first sample path because we decide to hold the asset at time 3 (based on our approximate value function) after which the price of the option turns out to be worth 0.05. Discounted, this is worth 0.0475. For sample path 2, the option proves to be worth 0.07 which discounts back to 0.0665 (we decided to hold at time 3, and the option was worth

Regression data				
	Independent variables			Dependent variable
Outcome	X_1	X_2	X_3	Y
1	1.08	1.17	1.3689	0.05
2	1.12	1.17	1.3689	0.07
3	1.08	1.22	1.4884	0.00
4	1.12	1.18	1.3924	0.05
5	1.15	1.10	1.2100	0.00
6	1.22	1.23	1.5129	0.03
7	1.44	1.13	1.2769	0.01
8	1.18	1.21	1.4641	0.00
9	1.11	1.09	1.1881	0.10
10	1.14	1.18	1.3924	0.00

Table 18.5 The data table for our regression at time 3

0.07 at time 4). For sample path 5 the option is worth 0.10 because we decided to exercise at time 3.

Regardless of which way we compute the value of the problem at time 3, the remainder of the procedure is the same. We have to construct the independent variables "Y" and regress them against our observations of the value of the option at time 3 using the price history (p_1, p_2). Our only change in methodology would occur at time 1 where we would have to use a different model (because we do not have a price at time 0).

Playing "lose tic-tac-toe"

The game of "lose tic-tac-toe" is the same as the familiar game of tic-tac-toe, with the exception that now you are trying to make the other person get three in a row. This nice twist on the popular children's game provides the setting for our next use of regression methods in approximate dynamic programming.

Unlike our exercise in pricing options, representing a tic-tac-toe board requires capturing a discrete state. Assume the cells in the board are numbered left to right, top to bottom as shown in figure 18.8a. Now consider the board in figure 18.8b. We can represent the state of the board after the t^{th} play using

$$\begin{aligned} S_{ti} &= \begin{cases} 1 & \text{if cell } i \text{ contains an "X,"} \\ 0 & \text{if cell } i \text{ is blank,} \\ -1 & \text{if cell } i \text{ contains an "O,"} \end{cases} \\ S_t &= (S_{ti})_{i=1}^9. \end{aligned}$$

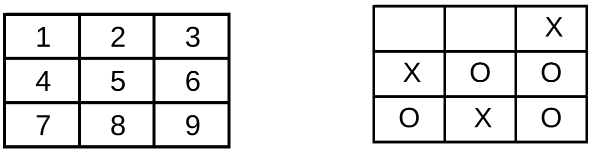
We see that this simple problem has up to $3^9 = 19,683$ states. While many of these states will never be visited, the number of possibilities is still quite large, and seems to overstate the complexity of the game.

Rewards		
	Decision	
Outcome	Exercise	Hold
1	0.03	$0.04155 \times .95 =$ **0.03947**
2	0.03	$0.03662 \times .95 =$ **0.03479**
3	0.00	$0.02397 \times .95 =$ **0.02372**
4	0.02	$0.03346 \times .95 =$ **0.03178**
5	**0.10**	$0.05285 \times .95 = 0.05021$
6	0.00	$0.00414 \times .95 =$ **0.00394**
7	**0.07**	$0.00899 \times .95 = 0.00854$
8	0.00	$0.01610 \times .95 =$ **0.01530**
9	**0.11**	$0.06032 \times .95 = 0.05731$
10	0.02	$0.03099 \times .95 =$ **0.02944**

Table 18.6 The payout if we exercise at time 3, and the expected value of holding based on our approximation. The best decision is indicated in bold.

We quickly realize that what is important about a game board is not the status of every cell as we have represented it. For example, rotating the board does not change a thing, but it does represent a different state. Also, we tend to focus on strategies (early in the game when it is more interesting) such as winning the center of the board or a corner. We might start defining variables (basis functions) such as

$$\begin{aligned}
\phi_1(S_t) &= \text{1 if there is an "X" in the center of the board, 0 otherwise,} \\
\phi_2(S_t) &= \text{The number of corner cells with an "X,"} \\
\phi_3(S_t) &= \text{The number of instances of adjacent cells with an "X" (horizontally,} \\
&\quad\ \text{vertically, or diagonally).}
\end{aligned}$$



18.8a

18.8b

Figure 18.8 Some tic-tac-toe boards. (18.8a) Our indexing scheme. (18.8b) Sample board.

There are, of course, numerous such functions we can devise, but it is unlikely that we could come up with more than a few dozen (if that) which appeared to be useful. It is important to realize that we do not need a value function to tell us to make obvious moves.

Once we form our basis functions, our value function approximation is given by

$$\overline{V}_t(S_t) = \sum_{f \in \mathcal{F}} \theta_{tf} \phi_f(S_t).$$

We note that we have indexed the parameters by time (the number of plays) since this might play a role in determining the value of the feature being measured by a basis function, but it is reasonable to try fitting a model where $\theta_{tf} = \theta_f$. We estimate the parameters θ by playing the game (and following some policy) after which we see if we won or lost. We let $Y^n = 1$ if we won the n^{th} game, 0 otherwise. This also means that the value function is trying to approximate the probability of winning if we are in a particular state.

We may play the game by using our value functions to help determine a policy. Another strategy, however, is simply to allow two people (ideally, experts) to play the game and use this to collect observations of states and game outcomes. This is an example of supervised learning. If we lack a "supervisor" then we have to depend on simple strategies combined with the use of slowly learned value function approximations. In this case, we also have to recognize that in the early iterations, we are not going to have enough information to reliably estimate the coefficients for a large number of basis functions.

18.4.2 A geometric view of basis functions*

For readers comfortable with linear algebra, we can obtain an elegant perspective on the geometry of basis functions. In section 3.7.1, we found the parameter vector θ for a regression model by minimizing the expected square of the errors between our model and a set of observations. Assume now that we have a "true" value function $V(s)$ which gives the value of being in state s, and let $p(s)$ be the probability of visiting state s. We wish to find the approximate value function that best fits $V(s)$ using a given set of basis functions $(\phi_f(s))_{f \in \mathcal{F}}$. If we minimize the expected square of the errors between our approximate model and the true value function, we would want to solve

$$\min_\theta F(\theta) = \sum_{s \in \mathcal{S}} p(s) \left(V(s) - \sum_{f \in \mathcal{F}} \theta_f \phi_f(s) \right)^2, \tag{18.21}$$

where we have weighted the error for state s by the probability of actually being in state s. Our parameter vector θ is unconstrained, so we can find the optimal value by taking the derivative and setting this equal to zero. Differentiating with respect to $\theta_{f'}$ gives

$$\frac{\partial F(\theta)}{\partial \theta_{f'}} = -2 \sum_{s \in \mathcal{S}} p(s) \left(V(s) - \sum_{f \in \mathcal{F}} \theta_f \phi_f(s) \right) \phi_{f'}(s).$$

Setting the derivative equal to zero and rearranging gives

$$\sum_{s \in \mathcal{S}} p(s) V(s) \phi_{f'}(s) = \sum_{s \in \mathcal{S}} p(s) \sum_{f \in \mathcal{F}} \theta_f \phi_f(s) \phi_{f'}(s). \tag{18.22}$$

At this point, it is much more elegant to revert to matrix notation. Define an $|\mathcal{S}| \times |\mathcal{S}|$ diagonal matrix D where the diagonal elements are the state probabilities $p(s)$, as follows

$$D = \begin{pmatrix} p(1) & 0 & & 0 \\ 0 & p(2) & & 0 \\ \vdots & 0 & \cdots & \vdots \\ 0 & \vdots & & p(|\mathcal{S}|) \end{pmatrix}.$$

Let V be the column vector giving the value of being in each state

$$V = \begin{pmatrix} V(1) \\ V(2) \\ \vdots \\ V(|\mathcal{S}|) \end{pmatrix}.$$

Finally, let Φ be an $|\mathcal{S}| \times |\mathcal{F}|$ matrix of the basis functions given by

$$\Phi = \begin{pmatrix} \phi_1(1) & \phi_2(1) & & \phi_{|\mathcal{F}|}(1) \\ \phi_1(2) & \phi_2(2) & & \phi_{|\mathcal{F}|}(2) \\ \vdots & \vdots & \cdots & \vdots \\ \phi_1(|\mathcal{S}|) & \phi_2(|\mathcal{S}|) & & \phi_{|\mathcal{F}|}(|\mathcal{S}|) \end{pmatrix}.$$

Recognizing that equation (18.22) is for a particular feature f', with some care it is possible to see that equation (18.22) for all features is given by the matrix equation

$$\Phi^T DV = \Phi^T D\Phi\theta. \tag{18.23}$$

It helps to keep in mind that Φ is an $|\mathcal{S}| \times |\mathcal{F}|$ matrix, D is an $|\mathcal{S}| \times |\mathcal{S}|$ diagonal matrix, V is an $|\mathcal{S}| \times 1$ column vector, and θ is an $|\mathcal{F}| \times 1$ column vector. The reader should carefully verify that (18.23) is the same as (18.22).

Now, pre-multiply both sides of (18.23) by $(\Phi^T D\Phi)^{-1}$. This gives us the optimal value of θ as

$$\theta = (\Phi^T D\Phi)^{-1}\Phi^T DV. \tag{18.24}$$

This equation is closely analogous to the normal equations of linear regression, given by equation (3.52), with the only difference being the introduction of the scaling matrix D which captures the probability that we are going to visit a state.

Now, pre-multiply both sides of (18.24) by Φ, which gives

$$\Phi\theta = \overline{V} = \Phi(\Phi^T D\Phi)^{-1}\Phi^T DV.$$

$\Phi\theta$ is, of course, our approximation of the value function, which we have denoted by $\overline{V}$. This, however, is the best possible value function given the set of functions $\phi = (\phi_f)_{f\in\mathcal{F}}$. If the vector ϕ formed a complete basis over the space formed by the value function $V(s)$ and the state space $\mathcal{S}$, then we would obtain $\Phi\theta = \overline{V} = V$. Since this is generally not the case, we can view $\overline{V}$ as the nearest point projection (where "nearest" is defined as a weighted measure using the state probabilities $p(s)$) onto the space formed by the basis functions. In fact, we can form a projection operator Π defined by

$$\Pi = \Phi(\Phi^T D\Phi)^{-1}\Phi^T D$$

so that $\overline{V} = \Pi V$ is the value function closest to V that can be produced by the set of basis functions.

This discussion brings out the geometric view of basis functions (and at the same time, the reason why we use the term "basis function"). There is an extensive literature on basis functions that has evolved in the approximation literature.

18.5 APPROXIMATE POLICY ITERATION

One of the most important tools in the toolbox for approximate dynamic programming is approximate policy iteration. This algorithm is neither simpler nor more elegant than approximate value iteration, but it can offer convergence guarantees while using linear models to approximate the value function.

In this section we review several flavors of approximate policy iteration, including

a) Finite horizon problems using lookup tables.

b) Finite horizon problems using basis functions.

c) Infinite horizon problems using basis functions.

Finite horizon problems allow us to obtain Monte Carlo estimates of the value of a policy by simulating the policy until the end of the horizon. Note that a "policy" here always refers to decisions that are determined by value function approximations. We use the finite horizon setting to illustrate approximating value function approximations using lookup tables and basis functions, which allows us to highlight the strengths and weaknesses of the transition to basis functions.

We then present an algorithm based on least squares temporal differences (LSTD) and contrast the steps required for finite horizon and infinite horizon problems when using basis functions.

18.5.1 Finite horizon problems using lookup tables

A fairly general purpose version of an approximate policy iteration algorithm is given in figure 18.9 for an infinite horizon problem. This algorithm helps to illustrate the choices that can be made when designing a policy iteration algorithm in an approximate setting. The algorithm features three nested loops. The innermost loop steps forward and backward in time from an initial state $S^{n,0}$. The purpose of this loop is to obtain an estimate of the value of a path. Normally, we would choose T large enough so that γ^T is quite small (thereby approximating an infinite path). The next outer loop repeats this process M times to obtain a statistically reliable estimate of the value of a policy (determined by $\overline{V}^{\pi,n}$). The third loop, representing the outer loop, performs policy updates (in the form of updating the value function). In a more practical implementation, we might choose states at random rather than looping over all states.

Readers should note that we have tried to index variables in a way that shows how they are changing (do they change with outer iteration n? inner iteration m? the forward look-ahead counter t?). This does not mean that it is necessary to store, for example, each state or decision for every n, m, and t. In an actual implementation, the software should be designed to store only what is necessary.

We can create different variations of approximate policy iteration by our choice of parameters. First, if we let $T \rightarrow \infty$, we are evaluating a true infinite horizon policy. If

```
Step 0. Initialization:
    Step 0a. Initialize $\overline{V}^{\pi,0}$.
    Step 0b. Set a look-ahead parameter $T$ and inner iteration counter $M$.
    Step 0c. Set $n = 1$.
Step 1. Sample a state $S_0^n$ and then do:
Step 2. Do for $m = 1, 2, \dots, M$:
    Step 3. Choose a sample path $\omega^m$ (a sample realization over the lookahead horizon $T$).
    Step 4. Do for $t = 0, 1, \dots, T$:
        Step 4a. Compute
            $$a_t^{n,m} = \arg\max_{a_t \in \mathcal{A}_t^{n,m}} \big( C(S_t^{n,m}, a_t) + \gamma \overline{V}^{\pi,n-1}(S^{M,a}(S_t^{n,m}, a_t)) \big).$$
        Step 4b. Compute
            $$S_{t+1}^{n,m} = S^M(S_t^{n,m}, a_t^{n,m}, W_{t+1}(\omega^m)).$$
    Step 5. Initialize $\hat{v}_{T+1}^{n,m} = 0$.
    Step 6: Do for $t = T, T-1, \dots, 0$:
        Step 6a: Accumulate $\hat{v}^{n,m}$:
            $$\hat{v}_t^{n,m} = C(S_t^{n,m}, a_t^{n,m}) + \gamma \hat{v}_{t+1}^{n,m}.$$
        Step 6b: Update the approximate value of the policy:
            $$\bar{v}^{n,m} = (\frac{m-1}{m})\bar{v}^{n,m-1} + \frac{1}{m}\hat{v}_0^{n,m}.$$
Step 8. Update the value function at $S^n$:
    $$\overline{V}^{\pi,n} = (1 - \alpha_{n-1})\bar{v}^{n-1} + \alpha_{n-1}\hat{v}_0^{n,M}.$$
Step 9. Set $n = n + 1$. If $n < N$, go to Step 1.
Step 10. Return the value functions $(\overline{V}^{\pi,N})$.
```

Figure 18.9 A policy iteration algorithm for infinite horizon problems

we simultaneously let $M \to \infty$, then $\bar{v}^n$ approaches the exact, infinite horizon value of the policy π determined by $\overline{V}^{\pi,n}$. Thus, for $M = T = \infty$, we have a Monte Carlo-based version of exact policy iteration.

We can choose a finite value of T that produces values $\hat{v}^{n,m}$ that are close to the infinite horizon results. We can also choose finite values of M, including $M = 1$. When we use finite values of M, this means that we are updating the policy before we have fully evaluated the policy. This variant is known in the literature as *optimistic policy iteration* because rather than wait until we have a true estimate of the value of the policy, we update the policy after each sample (presumably, although not necessarily, producing a better policy). We may also think of this as a form of partial policy evaluation, not unlike the hybrid value/policy iteration described in section 14.6.

```
Step 0. Initialization:
    Step 0a. Fix the basis functions $\phi_f(s)$.
    Step 0b. Initialize $\theta_{tf}^{\pi,0}$ for all $t$. This determines the policy we simulate in the inner loop.
    Step 0c. Set $n = 1$.
    Step 1. Sample an initial starting state $S_0^n$:
        Step 2. Initialize $\theta^{n,0}$ (if $n > 1$, use $\theta^{n,0} = \theta^{n-1}$), which is used to estimate the value of policy
                $\pi$ produced by $\theta^{pi,n}$. $\theta^{n,0}$ is used to approximate the value of following policy $\pi$ determined
                by $\theta^{\pi,n}$.
        Step 3. Do for $m = 1, 2, \ldots, M$:
            Step 4. Choose a sample path $\omega^m$.
            Step 5. Do for $t = 0, 1, \ldots, T$:
                Step 5a. Compute
                    $a_t^{n,m} = \arg\max_{a_t \in \mathcal{A}_t^{n,m}} \left( C(S_t^{n,m}, a_t) + \gamma \sum_f \theta_{tf}^{\pi,n-1} \phi_f(S^{M,a}(S_t^{n,m}, a_t)) \right).$
                Step 5b. Compute
                    $S_{t+1}^{n,m} = S^M(S_t^{n,m}, a_t^{n,m}, W_{t+1}(\omega^m)).$
            Step 6. Initialize $\hat{v}_{T+1}^{n,m} = 0$.
            Step 7: Do for $t = T, T-1, \ldots, 0$:
                    $\hat{v}_t^{n,m} = C(S_t^{n,m}, a_t^{n,m}) + \gamma \hat{v}_{t+1}^{n,m}.$
            Step 8. Update $\theta_t^{n,m-1}$ using recursive least squares to obtain $\theta_t^{n,m}$ (see section 3.8).
Step 9. Set $n = n + 1$. If $n < N$, go to Step 1.
Step 10. Return the value functions $(\overline{V}^{\pi,N})$.
```

Figure 18.10 A policy iteration algorithm for finite horizon problems using basis functions.

18.5.2 Finite horizon problems using basis functions

The simplest demonstration of approximate policy iteration using basis functions is in the setting of a finite horizon problem. Figure 18.10 provides an adaption of the algorithm using lookup tables when we are using basis functions. There is an outer loop over n where we fix the policy using

$$A_t^\pi(S_t) = \arg\max_a \left(C(S_t, a) + \gamma \sum_f \theta_{tf}^{\pi,n} \phi_f(S_t) \right). \tag{18.25}$$

We are assuming that the basis functions are not themselves time-dependent, although they depend on the state variable S_t which, of course, is time dependent. The policy is determined by the parameters $\theta_{tf}^{\pi,n}$.

We update the policy $A_t^\pi(s)$ by performing repeated simulations of the policy in an inner loop that runs $m = 1, \ldots, M$. Within this inner loop, we use recursive least squares to update a parameter vector $\theta_{tf}^{n,m}$. This step replaces step 6b in figure 18.9.

If we let $M \to \infty$, then the parameter vector $\theta_t^{n,M}$ approaches the best possible fit for the policy $A_t^\pi(s)$ determined by $\theta^{\pi,n-1}$. However, it is very important to realize that this is not equivalent to performing a perfect evaluation of a policy using a lookup table representation. The problem is that (for discrete states), lookup tables have the *potential* for perfectly approximating a policy, whereas this is not generally true when we use basis functions. If we have a poor choice of basis functions, we may be able find the best possible value of $\theta^{n,m}$ as m goes to infinity, but we may still have a terrible approximation of the policy produced by $\theta^{\pi,n-1}$.

18.5.3 LSTD for infinite horizon problems using basis functions

We have built the foundation for approximate policy iteration using lookup tables and basis functions for finite horizon problems. We now make the transition to infinite horizon problems using basis functions, where we introduce the dimension of projecting contributions over an infinite horizon. There are several ways of accomplishing this (see section 17.1.2). We use least squares temporal differencing, since it represents the most natural extension of classical policy iteration for infinite horizon problems.

To begin, we let a sample realization of a one-period contribution, given state S^m, action a^m and random information W^{m+1} be given by

$$\hat{C}^m = C(S^m, a^m, W^{m+1}).$$

As in the past, we let $\phi^m = \phi(S^m)$ be the column vector of basis functions evaluated at state S^m. We next fix a policy which chooses actions greedily based on a value function approximation given by $\overline{V}^n(s) = \sum_f \theta^n_f \phi_f(s)$ (see equation (18.25)). Imagine that we have simulated this policy over a set of iterations $i = (0, 1, \ldots, m)$, giving us a sequence of contributions $\hat{C}^i,\ i = 1, \ldots, m$. Drawing on the foundation provided in section 17.3, we can use standard linear regression to estimate θ^m using

$$\theta^m = \left[\frac{1}{1+m}\sum_{i=0}^{m} \phi_i(\phi^i - \gamma\phi^{i+1})^T\right]^{-1} \left[\frac{1}{1+m}\sum_{i=1}^{m} \phi^i \hat{C}^i \phi^i\right]. \tag{18.26}$$

As a reminder, the term $\phi^i - \gamma\phi^{i+1}$ can be viewed as a simulated, sample realization of $I - \gamma P^\pi$, projected into the feature space. Just as we would use $(I - \gamma P^\pi)^{-1}$ in our basic policy iteration to project the infinite-horizon value of a policy π (for a review, see section 14.5), we are using the term

$$\left[\frac{1}{1+m}\sum_{i=0}^{m} \phi_i(\phi^i - \gamma\phi^{i+1})^T\right]^{-1}$$

to produce an infinite-horizon estimate of the feature-projected contribution

$$\left[\frac{1}{1+m}\sum_{i=1}^{m} \phi^i \hat{C}^i \phi^i\right].$$

Equation (18.26) requires solving a matrix inverse for every observation. It is much more efficient to use recursive least squares, which is done by using

$$\begin{aligned}
\epsilon^m &= \hat{C}^m - (\phi^m - \gamma\phi^{m+1})^T \theta^{m-1}, & (18.27)\\
B^m &= B^{m-1} - \frac{B^m \phi^m (\phi^m - \gamma\phi^{m+1})^T B^{m-1}}{1 + (\phi^m - \gamma\phi^{m+1})^T B^{m-1}\phi^m}, & (18.28)\\
\theta^m &= \phi^{m-1} + \frac{\epsilon^m B^{m-1}\phi^m}{1 + (\phi^m - \gamma\phi^{m+1})^T B^{m-1}\phi^m}. & (18.29)
\end{aligned}$$

```
Step 0. Initialization:
    Step 0a. Initialize $\theta^0$.
    Step 0b. Set the initial policy:
        $A^\pi(s|\theta^0) = \arg\max_{a \in \mathcal{A}} \big(C(s,a) + \gamma\phi(S^M(s,a))^T\theta^0\big).$
    Step 0c. Set $n = 1$.
Step 1. Do for $n = 1, \ldots, N$.
    Step 2. Initialize $S_0^n$.
    Step 3. Do for $m = 0, 1, \ldots, M$:
        Step 4: Initialize $\theta^{n,m}$.
        Step 5: Sample $W^{m+1}$.
        Step 6: Do the following:
            Step 6a: Computing the action $a^{n,m} = A^\pi(S^m|\theta^{n-1})$.
            Step 6b: Compute the post-decision state $S^{a,m} = S^{M,a}(S^{n,m}, a^{n,m})$.
            Step 6c: Compute the next pre-decision state $S^{n,m+1} = S^M(S^{n,m}, a^{n,m}, W^{m+1})$.
            Step 6d: Compute the input variable $\phi(S^{n,m}) - \gamma\phi(S^{n,m+1})$ for equation (18.26).
        Step 7: Do the following:
            Step 7a: Compute the response variable $\hat{C}^m = C(S^{n,m}, a^{n,m}, W^{m+1})$.
            Step 7b: Compute $\theta^{n,m}$ using equation (18.26).
    Step 8: Update $\theta^n$ and the policy:
        $\theta^{n+1} = \theta^{n,m}$
        $A^{\pi,n+1}(s) = \arg\max_{a \in \mathcal{A}} \big(C(s,a) + \gamma\phi(S^M(s,a))\theta^{n+1}\big).$
Step 9. Return the $A^\pi(s|\theta^N)$ and parameter $\theta^N$.
```

Figure 18.11 Approximate policy iteration for infinite horizon problems using least squares temporal differencing.

Figure 18.11 provides a detailed summary of the complete algorithm. The algorithm has some nice properties if we are willing to assume that there is a vector θ^* such that the true value function $V(s) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(s)$ (admittedly, a pretty strong assumption). First, if the inner iteration limit M increases as a function of n so that the quality of the approximation of the policy gets better and better, then the overall algorithm will converge to the true optimal policy. Of course, this means letting $M \to \infty$, but from a practical perspective, it means that the algorithm can find a policy arbitrarily close to the optimal policy.

Second, the algorithm can be used with vector-valued and continuous actions (we normally use the notation x_t in this case). There are several features of the algorithm that allow this. First, computing the policy $A^\pi(s|\theta^n)$ requires solving a deterministic optimization problem. If we are using discrete actions, it means simply enumerating the actions and choosing the best one. If we have continuous actions, we need to solve a nonlinear programming problem. The only practical issue is that we may not be able to guarantee that the objective function is concave (or convex if we are minimizing). Second, note that we are using trajectory following (also known as on-policy training) in Step 6c, without an explicit exploration step. It can be very difficult implementing an exploration step for multidimensional decision vectors.

We can avoid exploration as long as there is enough variation in the states we visit that allows us to compute θ^m in equation (18.26). When we use lookup tables, we require exploration to guarantee that we eventually will visit every state infinitely often. When we use basis functions, we only need to visit states with sufficient diversity that we can estimate the parameter vector θ^m. In the language of statistics, the issue is one of *identification* (that is, the ability to estimate θ) rather than exploration. This is a much easier requirement to satisfy, and one of the major advantages of parametric models.

18.6 THE ACTOR-CRITIC PARADIGM

It is very popular in some communities to view approximate dynamic programming in terms of an "actor" and a "critic." Simply put, the actor is a policy and the critic is the mechanism for updating the policy, typically through updates to the value function approximation.

In this setting, a decision function that chooses a decision given the state is known as an *actor*. The process that determines the contribution (cost or reward) from a decision is known as the *critic*, from which we can compute a value function. The interaction of making decisions and updating the value function is referred to as an actor-critic framework. The slight change in vocabulary brings out the observation that the techniques of approximate dynamic programming closely mimic human behavior. This is especially true when we drop any notion of costs or contributions and simply work in terms of succeeding (or winning) and failing (or losing).

The policy iteration algorithm in figure 18.12 provides one illustration of the actor-critic paradigm. The decision function is equation (18.30), where $V^{\pi,n-1}$ determines the policy (in this case). This is the actor. Equation (18.31), where we update our estimate of the value of the policy, is the critic. We fix the actor for a period of time and perform repeated iterations where we try to estimate value functions given a particular actor (policy). From time to time, we stop and use our value function to modify our behavior (something critics like to do). In this case, we update the behavior by replacing V^π with our current $\overline{V}$.

In other settings, the policy is a rule or function that does not directly use a value function (such as V^π or $\overline{V}$). For example, if we are driving through a transportation network (or traversing a graph) the policy might be of the form "when at node i, go next to node j." As we update the value function, we may decide the right policy at node i is to traverse to node k. Once we have updated our policy, the policy itself does not directly depend on a value function.

Another example might arise when determining how much of a resource we should have on hand. We might solve the problem by maximizing a function of the form $f(x) = \beta_0 - \beta_1(x - \beta_2)^2$. Of course, β_0 does not affect the optimal quantity. We might use the value function to update β_0 and β_1. Once these are determined, we have a function that does not itself directly depend on a value function.

18.7 POLICY GRADIENT METHODS

Perhaps the cleanest illustration of the actor-critic framework arises when we parameterize both the value of being in a state as well as the policy. We use a standard strategy from the literature which uses Q-factors, and where the goal is to maximize the *average* contribution per time period (see section 14.7 for a brief introduction using the classical derivation based

```
Step 0. Initialization:
    Step 0a. Initialize $V_t^{\pi,0}$,  $t \in \mathcal{T}$.
    Step 0b. Set $n = 1$.
    Step 0c. Initialize $S_0^1$.
Step 1. Do for $n = 1, 2, \ldots, N$:
    Step 2. Do for $m = 1, 2, \ldots, M$:
        Step 3. Choose a sample path $\omega^m$.
        Step 4: Initialize $\hat{v}^m = 0$
        Step 5: Do for $t = 0, 1, \ldots, T$:
            Step 5a. Solve:
                $$a_t^{n,m} = \arg\max_{a_t \in \mathcal{A}_t^{n,m}} \left( C_t(S_t^{n,m}, a_t) + \gamma V_t^{\pi,n-1}(S^{M,a}(S_t^{n,m}, a_t)) \right) \qquad (18.30)$$
            Step 5b. Compute:
                $$S_t^{a,n,m} = S^{M,a}(S_t^{n,m}, a_t^{n,m})$$
                $$S_{t+1}^{n,m} = S^M(S_t^{a,n,m}, a^{n,m}, W_{t+1}(\omega^m)).$$
        Step 6. Do for $t = T-1, \ldots, 0$:
            Step 6a. Accumulate the path cost (with $\hat{v}_T^m = 0$)
                $$\hat{v}_t^m = C_t(S_t^{n,m}, a_t^m) + \gamma \hat{v}_{t+1}^m$$
            Step 6b. Update approximate value of the policy starting at time $t$:
                $$\overline{V}_{t-1}^{n,m} \leftarrow U^V(\overline{V}_{t-1}^{n,m-1}, S_{t-1}^{a,n,m}, \hat{v}_t^m) \qquad (18.31)$$
              where we typically use $\alpha_{m-1} = 1/m$.
    Step 7. Update the policy value function
        $$V_t^{\pi,n}(S_t^a) = \overline{V}_t^{n,M}(S_t^a) \quad \forall t = 0, 1, \ldots, T$$
Step 8. Return the value functions $(V_t^{\pi,N})_{t=1}^T$.
```

Figure 18.12 Approximate policy iteration using value function-based policies.

on transition matrices). Our presentation here represents only a streamlined sketch of an idea that is simple in principle but which involves some fairly advanced principles.

We assume that the Q-factors are parameterized using

$$\bar{Q}(s, a|\theta) = \sum_f \theta_f \phi_f(s, a),$$

The policy is represented using a function such as

$$A^\pi(s|\eta) = \frac{e^{\eta\phi(s,a)}}{\sum_{a'} \eta\phi(s, a')}.$$

This choice of policy has the important feature that the probability that an action is chosen is greater than zero. Also, $A^\pi(s|\eta)$ is differentiable in the policy parameter vector η.

In the language of actor-critic algorithms, $\bar{Q}(s, a|\theta)$ is an approximation of the critic parameterized by θ, while $A^\pi(s|\eta)$ is an approximate policy parameterized by η. We can update θ and η using standard stochastic gradient methods. We begin by defining

$$\begin{aligned}\psi_\theta(s,a) &= \nabla_\theta \ln \pi_\theta(a|s)\\ &= \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)}.\end{aligned}$$

Since we are maximizing the average reward per time period, we begin by estimating the average reward per time period using

$$\bar{c}^{n+1} = (1-\alpha_n)\bar{c}^n + \alpha_n C(S^{n+1}, a^{n+1}).$$

We then compute the temporal difference in terms of the difference between the contribution of a state-action pair, and the average contribution, using

$$\delta^n = C(S^n, a^n) - \bar{c}^n + (\theta^n)^T \phi_{\theta^n}(S^{n+1}, a^{n+1}) - (\theta^n)^T \phi_{\theta^n}(S^n, a^n).$$

Assume we are using a TD(λ) updating procedure where we assume $0 < \lambda < 1$. We compute the eligibility trace using

$$Z^{n+1} = \lambda Z^n + \phi_{\theta^n}(S^{n+1}, a^{n+1})$$

We can now present the updating equations for the actor (the policy) and the critic (the Q-factors) in a simple and compact way. The actor update is given by

$$\eta^{n+1} = \eta^n - \beta^n \Gamma(\theta^n)(\theta^n)^T \phi_{\theta^n}(S^{n+1}, a^{n+1})\psi_{\theta^n}(S^{n+1}, a^{n+1}). \tag{18.32}$$

The critic update is given by

$$\theta^{n+1} = \theta^n + \alpha_n \delta^n Z^n. \tag{18.33}$$

Equations (18.32) and (18.33) provide an elegant and compact illustration of an actor-critic updating equation, where both the value function and the policy are approximated using parametric models. This method is likely to be of limited practical value, largely because of the form of the policy which requires a positive probability that any action may be chosen.

18.8 THE LINEAR PROGRAMMING METHOD USING BASIS FUNCTIONS

In section 14.8, we showed that the determination of the value of being in each state can be found by solving the following linear program

$$\min_v \sum_{s\in\mathcal{S}} \beta_s v(s) \tag{18.34}$$

subject to

$$v(s) \geq C(s,x) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,x)v(s') \quad \text{for all } s \text{ and } x. \tag{18.35}$$

The problem with this formulation arises because it requires that we enumerate the state space to create the value function vector $(v(s))_{s\in\mathcal{S}}$. Furthermore, we have a constraint for each state-action pair, a set that will be huge even for relatively small problems.

We can partially solve this problem by replacing the discrete value function with a regression function such as

$$\overline{V}(s|\theta) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(s).$$

where $(\phi_f)_{f\in\mathcal{F}}$ is an appropriately designed set of basis functions. This produces a revised linear programming formulation

$$\min_{\theta} \sum_{s \in \mathcal{S}} \beta_s \sum_{f \in \mathcal{F}} \theta_f \phi_f(s)$$

subject to:

$$v(s) \quad \geq \quad C(s,x) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,x) \sum_{f \in \mathcal{F}} \theta_f \phi_f(s') \quad \text{for all } s \text{ and } x.$$

This is still a linear program, but now the decision variables are $(\theta_f)_{f\in\mathcal{F}}$ instead of $(v(s))_{s\in\mathcal{S}}$. Note that rather than use a stochastic iterative algorithm, we obtain θ directly by solving the linear program.

We still have a problem with a huge number of constraints. Since we no longer have to determine $|\mathcal{S}|$ decision variables (in (18.34)-(18.35) the parameter vector $(v(s))_{s\in\mathcal{S}}$ represents our decision variables), it is not surprising that we do not actually need all the constraints. One strategy that has been proposed is to simply choose a random sample of states and actions. Given a state space $\mathcal{S}$ and set of actions (decisions) $\mathcal{X}$, we can randomly choose states and actions to create a smaller set of constraints.

Some care needs to be exercised when generating this sample. In particular, it is important to generate states roughly in proportion to the probability that they will actually be visited. Then, for each state that is generated, we need to randomly sample one or more actions. The best strategy for doing this is going to be problem-dependent.

This technique has been applied to the problem of managing a network of queues. Figure 18.13 shows a queueing network with three servers and eight queues. A server can serve only one queue at a time. For example, server A might be a machine that paints components one of three colors (say, red, green, and blue). It is best to paint a series of parts red before switching over to blue. There are customers arriving exogenously (denoted by the arrival rates λ_1 and λ_2). Other customers arrive from other queues (for example, departures from queue 1 become arrivals to queue 2). The problem is to determine which queue a server should handle after each service completion.

If we assume that customers arrive according to a Poisson process and that all servers have negative exponential service times (which means that all processes are memoryless), then the state of the system is given by

$$S_t = R_t = (R_{ti})_{i=1}^{8},$$

where R_{ti} is the number of customers in queue i. Let $\mathcal{K} = \{1,2,3\}$ be our set of servers, and let a_t be the attribute vector of a server given by $a_t = (k, q_t)$, where k is the identity of the server and q_t is the queue being served at time t. Each server can only serve a subset of queues (as shown in figure 18.13). Let $\mathcal{D} = \{1,2,\ldots,8\}$ represent a decision to serve a particular queue, and let $\mathcal{D}_a$ be the decisions that can be used for a server with attribute a. Finally, let $x_{tad} = 1$ if we decide to assign a server with attribute a to serve queue $d \in \mathcal{D}_a$.

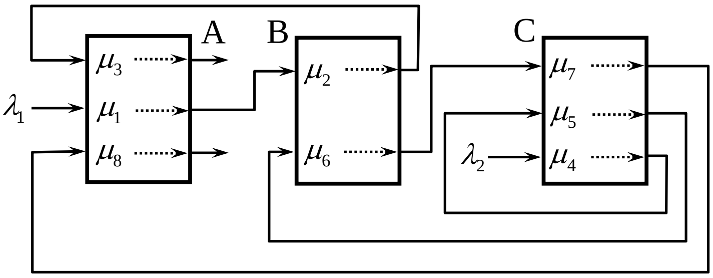


Figure 18.13 Queueing network with three servers serving a total of eight queues, two with exogenous arrivals (λ) and six with arrivals from other queues (from de Farias & Van Roy (2003)).

Policy	Cost
ADP	33.37
Longest	45.04
FIFO	45.71

Table 18.7 Average cost estimated using simulation (from de Farias & Van Roy (2003)).

The state space is effectively infinite (that is, too large to enumerate). But we can still sample states at random. Research has shown that it is important to sample states roughly in proportion to the probability they are visited. We do not know the probability a state will be visited, but it is known that the probability of having a queue with r customers (when there are Poisson arrivals and negative exponential servers) follows a geometric distribution. For this reason, it was chosen to sample a state with $r = \sum_i R_{ti}$ customers with probability $(1-\gamma)\gamma^r$, where γ is a discount factor (a value of 0.95 was used).

Further complicating this problem class is that we also have to sample actions. Let $\mathcal{X}$ be the set of all feasible values of the decision vector x. The number of possible decisions for each server is equal to the number of queues it serves, so the total number of values for the vector x is $3 \times 2 \times 3 = 18$. In the experiments for this illustration, only 5,000 states were sampled (in portion to $(1-\gamma)\gamma^r$), but all the actions were sampled for each state, producing 90,000 constraints.

Once the value function is approximated, it is possible to simulate the policy produced by this value function approximation. The results were compared against two myopic policies: serving the longest queue, and first-in, first-out (that is, serve the customer who had arrived first). The costs produced by each policy are given in table 18.7, showing that the ADP-based strategy significantly outperforms these other policies.

Considerably more numerical work is needed to test this strategy on more realistic systems. For example, for systems that do not exhibit Poisson arrivals or negative exponential service times, it is still possible that sampling states based on geometric distributions may work quite well. More problematic is the rapid growth in the feasible region $\mathcal{X}$ as the number of servers, and queues per server, increases.

An alternative to using constraint sampling is an advanced technique known as column generation. Instead of generating a full linear program which enumerates all decisions (that is, $v(s)$ for each state), and all constraints (equation (18.35)), it is possible to generate sequences of larger and larger linear programs, adding rows (constraints) and columns (decisions) as needed. These techniques are beyond the scope of our presentation, but readers need to be aware of the range of techniques available for this problem class.

18.9 APPROXIMATE POLICY ITERATION USING KERNEL REGRESSION*

We build on the foundation provided in section 17.6 that describes the use of kernel regression in the context of least squares temporal difference (LSTD) learning. As we have done earlier, we let the one-period contribution be given by

$$\hat{C}^m = C(S^{n,m}, a^{n,m}, W^{m+1}).$$

Let $S^{a,i},\ i = 1, \dots, m$ be the sample-path of post-decision states produced by following a policy. Let $k(S^{a,i}, S^{a,j})$ be the normalized kernel function given by

$$k(S^{a,i}, S^{a,j}) = \frac{K_h(S^{a,i}, S^{a,j})}{\sum_{i=0}^{m-1} K_h(S^{a,i}, S^{a,j})},$$

which means that $\sum_{i=0}^{m-1} k(S^{a,i}, S^{a,j}) = 1$. Then, let $P^{\pi,n}$ be a $M \times M$ matrix where the $(i,j)th$ entry is given by

$$P^{\pi,n}_{i,j} = k(S^{a,i-1}, S^{a,j}).$$

By construction, $P^{\pi,n}$ is a stochastic matrix (its rows sum to 1), which means that $I - \gamma P^{\pi,n}$ is invertible.

Define the kernel-based approximation of Bellman's operator for a fixed policy $\hat{M}^{\pi,m}$ from the sample path of post-decision states $S^{a,0}, \dots, S^{a,m+1}$ using

$$\hat{M}^{\pi,m} V(s) = \sum_{i=0}^{m-1} k(S^{a,i}, s)(\hat{C}^i + \gamma V(S^{M,a}(S^i, a^i, W^{i+1})).$$

We would like to find the fixed point of the kernel-based Bellman equation defined by

$$\begin{aligned} \hat{V}^\pi &= \hat{M}^{\pi,m} \hat{V}^\pi \\ &= P^\pi [c^\pi + \gamma \hat{V}^\pi] \\ &= [I - \gamma P^\pi]^{-1} c^\pi. \end{aligned}$$

We can avoid the matrix inversion by using a value iteration approximation

$$\hat{V}^{\pi,k+1} = P^\pi (c^\pi + \gamma \hat{V}^{\pi,i}).$$

The vector $\hat{V}^\pi$ has an element $\hat{V}^\pi(S^{a,i})$ for each of the (post-decision) states $S^{a,i}$ that we have visited. We then extrapolate from this vector of calculated values for the states we have visited, giving us the continuous function

$$\overline{V}^\pi(s) = \sum_{i=0}^{m-1} k(S^{a,i}, s) \left(\hat{C}^i + \gamma \hat{V}^\pi(S^{a,i+1}) \right).$$

This approximation forms the basis of our approximate policy iteration. The full algorithm is given in figure 18.14.

```
Step 0. Initialization:
    Step 0a. Initialize the policy $A^\pi(s)$.
    Step 0b. Choose the kernel function $K_h(s, s')$.
    Step 0b. Set $n = 1$.
Step 1. Do for $n = 1, \ldots, N$.
    Step 2. Choose an initial state $S_0^n$.
    Step 3. Do for $m = 0, 1, \ldots, M$:
        Step 4: Let $a^{n,m} = A^{\pi,n}(S^{n,m})$.
        Step 5: Sample $W^{m+1}$.
        Step 6: Compute the post-decision state $S^{a,m} = S^{M,a}(S^{n,m}, a^{n,m})$ and the next state
            $S^{m+1} = S^M(S^{n,m}, a^{n,m}, W^{m+1})$.
    Step 7: Let $c^\pi$ be a vector of dimensionality $M$ with element $\hat{C}^m = C(S^{n,m}, a^{n,m}, W^{m+1}),\ m =$
        $1, \ldots, M$.
    Step 8: Let $P^{\pi,n}$ be a $M \times M$ matrix where the $(i,j)th$ entry is given by $K_h(S^{a,i-1}, S^{a,j})$ for
        $i, j \in \{1, \ldots, m\}$.
    Step 9: Solve for $\hat{v}^n = (I - \gamma P^{\pi,n})^{-1}$, where $\hat{v}^n$ is an $m$-dimensional vector with $ith$ element
        $\hat{v}^n(S^{a,i})$ for $i = 1, \ldots, m$. This can be approximated using value iteration.
    Step 10: Let $\overline{V}^n(s) = \sum_{i=0}^{m-1} K_h(S^{a,i}, s)(\hat{C}^i + \gamma \hat{v}^n(S^{a,i}))$ be our kernel-based value function
        approximation.
    Step 11: Update the policy:
            $A^{\pi,n+1}(s) = \arg\max_a \left(C(s,a) + \gamma \overline{V}^n(S^{M,a}(s,a))\right.$.
Step 12. Return the $A^{\pi,N}(s)$ and parameter $\theta^N$.
```

Figure 18.14 Approximate policy iteration using least squares temporal differencing and kernel regression.

18.10 FINITE HORIZON APPROXIMATIONS FOR STEADY-STATE APPLICATIONS

It is easy to assume that if we have a problem with stationary data (that is, all random information is coming from a distribution that is not changing over time), then we can solve the problem as an infinite horizon problem, and use the resulting value function to produce a policy that tells us what to do in any state. If we can, in fact, find the optimal value function for every state, this is true.

There are many applications of infinite horizon models to answer policy questions. Do we have enough doctors? What if we increase the buffer space for holding customers in a queue? What is the impact of lowering transaction costs on the amount of money a mutual fund holds in cash? What happens if a car rental company changes the rules allowing rental offices to give customers a better car if they run out of the type of car that a customer reserved? These are all dynamic programs controlled by a constraint (the size of a buffer or the number of doctors), a parameter (the transaction cost), or the rules governing the physics of the problem (the ability to substitute cars). We may be interested in understanding the behavior of such a system as these variables are adjusted. For infinite horizon problems that are too complex to solve exactly, ADP offers a way to approximate these solutions.

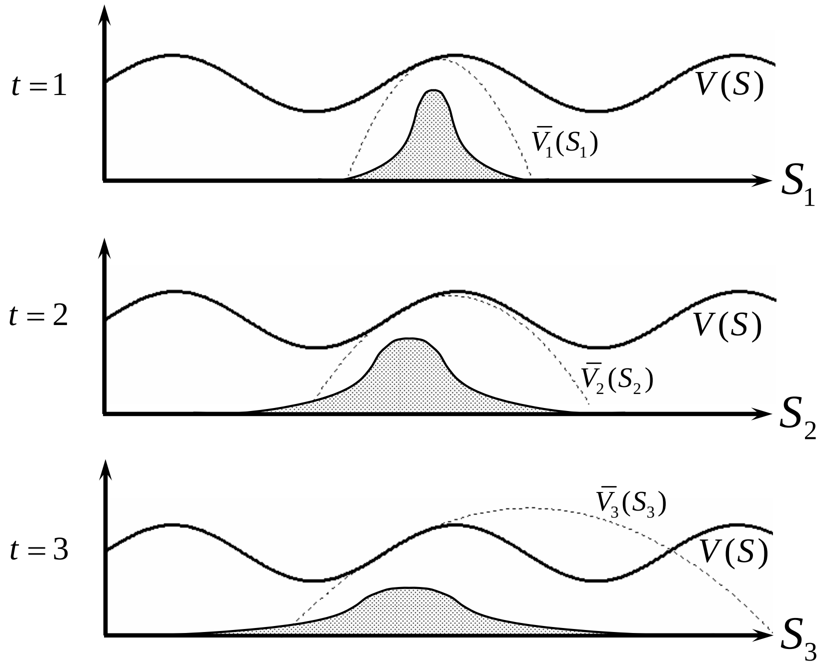


Figure 18.15 Exact value function (sine curve) and value function approximations for $t = 1, 2, 3$, which change with the probability distribution of the states that we can reach from S_0.

Infinite horizon models also have applications in operational settings. Assume that we have a problem governed by stationary processes. We could solve the steady-state version of the problem, and use the resulting value function to define a policy that would work from any starting state. This works if we have, in fact, found at least a close approximation of the optimal value function for any starting state. However, if you have made it this far in this book, then that means you are interested in working on problems where the optimal value function cannot be found for all states. Typically, we are forced to approximate the value function, and it is always the case that we do the best job of fitting the value function around states that we visit most of the time.

When we are working in an operational setting, then we start with some known initial state S_0. From this state, there are a range of "good" decisions, followed by random information, that will take us to a set of states S_1 that is typically heavily influenced by our starting state. Figure 18.15 illustrates the phenomenon. Assume that our true, steady-state value function approximation looks like the sine function. At time $t = 1$, the probability distribution of the state S_t that we can reach is shown as the shaded area. Assume that we have chosen to fit a quadratic function of the value function, using observations of S_t that we generate through Monte Carlo sampling. We might obtain the dotted curve labeled as $\overline{V}_1(S_1)$, which closely fits the true value function around the states S_1 that we have observed.

For times $t = 2$ and $t = 3$, the distribution of states S_2 and S_3 that we actually observe grows wider and wider. As a result, the best fit of a quadratic function spreads as well. So, even though we have a steady-state problem, the best value function approximation depends on the initial state S_0 and how many time periods into the future that we are projecting. Such problems are best modeled as finite horizon problems, but only because we are forced to approximate the problem.

18.11 BIBLIOGRAPHIC NOTES

Section 18.2 - Approximate value iteration using lookup tables encompasses the family of algorithms that depend on an approximation of the value of a future state to estimate the value of being in a state now, which includes Q-learning and temporal-difference learning. These methods represent the foundation of approximate dynamic programming and reinforcement learning.

Section 18.4 - The problems with the use of linear models in the context of approximate value iteration (TD learning) are well known in the research literature. Good discussions of these issues are found in Bertsekas & Tsitsiklis (1996), Tsitsiklis et al. (1997), Baird (1995) and Precup et al. (2001), to name a few.

Section 18.5 - Bradtke & Barto (1996) first introduced least squares temporal differencing, which is a way of approximating the one-period contribution using a linear model, and then projecting the infinite horizon performance. Lagoudakis & Parr (2003) describes the least squares policy iteration algorithm (LSPI) which uses a linear model to approximate the Q-factors, which is then imbedded in a model-free algorithm.

Section 18.6 - There is a long history of referring to policies as "actors" and value functions as "critics" (see, for example, Barto et al. (1983), Williams & Baird (1990), Bertsekas & Tsitsiklis (1996) and Sutton & Barto (1998)). Borkar & Konda (1997) and Konda & Borkar (1999) analyze actor-critic algorithms as an updating process with two time-scales, one for the inner iteration to evaluate a policy, and one for the outer iteration where the policy is updated. Konda & Tsitsiklis (2003) discusses actor-critic algorithms using linear models to represent both the actor and the critic, using bootstrapping for the critic. Bhatnagar et al. (2009) suggest several new variations of actor-critic algorithms, and proves convergence when both the actor and the critic use bootstrapping.

Section 18.7 - Policy gradient methods have received considerable attention in the reinforcement learning community. The material in this section is based on Konda & Tsitsiklis (2003), but we have provided only a streamlined presentation, and we urge readers to consult the original article before attempting to implement the equations given in this section. One of the earliest policy-gradient algorithms is given in Williams (1992). Marbach & Tsitsiklis (2001) provides gradient-based algorithms for optimizing Markov reward processes, which is a mathematically equivalent problem. Sutton et al. (2000) provides a version of a policy-gradient algorithm, but in a form which is difficult to compute. Sutton et al. (1983) compares several policy gradient algorithms. Szepesvari (2010) provides a recent summary of policy gradient algorithms.

Section 18.8 - Schweitzer & Seidmann (1985) describes the use of basis functions in the context of the linear programming method. The idea is further developed in Farias & Roy (2003) which also develops performance guarantees. Farias & Roy

(2001) investigates the use of constraint sampling and proves results on the number of samples that are needed.

Section 18.9 - This material is based on Ma & Powell (2010).

PROBLEMS

18.1 We are going to try again to solve our asset selling problem, We assume we are holding a real asset and we are responding to a series of offers. Let $\hat{p}_t$ be the t^{th} offer, which is uniformly distributed between 500 and 600 (all prices are in thousands of dollars). We also assume that each offer is independent of all prior offers. You are willing to consider up to 10 offers, and your goal is to get the highest possible price. If you have not accepted the first nine offers, you must accept the 10^{th} offer.

(a) Write out the decision function you would use in a dynamic programming algorithm in terms of a Monte Carlo sample of the latest price and a current estimate of the value function.

(b) Write out the updating equations (for the value function) you would use after solving the decision problem for the t^{th} offer.

(c) Implement an approximate dynamic programming algorithm using *synchronous* state sampling. Using 1000 iterations, write out your estimates of the value of being in each state immediately after each offer. For this exercise, you will need to discretize prices for the purpose of approximating the value function. Discretize the value function in units of 5 dollars.

(d) From your value functions, infer a decision rule of the form "sell if the price is greater than $\bar{p}_t$."

CHAPTER 19

FORWARD ADP III: CONVEX FUNCTIONS

There is a genuinely vast range of problems that can be broadly described as resource allocation, where we are managing the flows of vaccines, blood, money, trucks, inventory and people. In many of these problems, the objective function is concave (convex if minimizing) in the resource vector, which means that the value function is concave. It turns out that concavity is a particularly valuable property when solving dynamic programs using value function approximations.

We are going to address the following problem classes:

- Scalar resource allocation problems - These arise often when managing the inventory of a single product, cash management, and energy storage problems (to name just a few).

- Multidimensional resource allocation problems - Once again,

Both of these problems can be solved using different models of information other than the resource state. Variations include:

- No side information.

- Side information, which is memoryless - For example while managing the energy in a battery to store excess power from a solar panel, we may be willing to model the solar energy at time t as being independent of the solar energy at time $t-1$. With a memoryless process, their is no post-decision information state, since all past information can be forgotten once we have made a decision.

- Side information which is first-order Markov - Now we allow the information at time t (such as prices, wind) to depend on the information state at time $t-1$. We note that the information state at time $t-1$ may depend on information that arrived before $t-1$. It also may be a general function of past information.

19.1 PIECEWISE LINEAR APPROXIMATIONS FOR SCALAR FUNCTIONS

There are many problems where we have to estimate the value of having a quantity R of some resource (where R is a scalar). We might want to know the value of having R dollars in a budget, R pieces of equipment, or R units of some inventory. R may be discrete or continuous, but we are going to focus on problems where R is either discrete or is easily discretized.

Assume now that we want to estimate a function $V(R)$ that gives the value of having R resources. There are applications where $V(R)$ increases or decreases monotonically in R. There are other applications where $V(R)$ is piecewise linear, concave (or convex) in R, which means the slopes of the function are monotonically decreasing (if the function is concave) or increasing (if it is convex). When the function (or the slopes of the function) is steadily increasing or decreasing, we would say that the function is *monotone*. If the function is increasing in the state variable, we might say that it is "monotonically increasing," or that it is *isotone* (although the latter term is not widely used). To say that a function is "monotone" can mean that it is monotonically increasing or decreasing.

Assume we have a function that is monotonically decreasing, which means that while we do not know the value function exactly, we know that $V(R+1) \leq V(R)$ (for scalar R). If our function is piecewise linear concave, then we will assume that $V(R)$ refers to the slope at R (more precisely, to the right of R). Assume our current approximation $\overline{V}^{n-1}(R)$ satisfies this property, and that at iteration n, we have a sample observation of $V(R)$ for $R = R^n$. If our function is piecewise linear concave, then $\hat{v}^n$ would be a sample realization of a derivative of the function. If we use our standard updating algorithm, we would write

$$\overline{V}^n(R^n) = (1-\alpha_{n-1})\overline{V}^{n-1}(R^n) + \alpha_{n-1}\hat{v}^n.$$

After the update, it is quite possible that our updated approximation no longer satisfies our monotonicity property. Below we review three strategies for maintaining monotonicity:

The leveling algorithm - A simple method that imposes monotonicity by simply forcing elements of the series which violate monotonicity to a larger or smaller value so that monotonicity is restored.

The SPAR algorithm - SPAR takes the points that violate monotonicity and simply averages them.

The CAVE algorithm - If there is a monotonicity violation after an update, CAVE simply expands the range of the function over which the update is applied.

The leveling algorithm and the SPAR algorithm enjoy convergence proofs, but the CAVE algorithm is the one that works the best in practice. We report on all three to provide a sense of algorithmic choices, and an enterprising reader may design a modification of one of the first two algorithms to incorporate the features of CAVE that make it work so well.

19.1.0.1 The leveling algorithm The leveling algorithm uses a simple updating logic that can be written as follows:

$$\overline{V}^n(y) = \begin{cases} (1-\alpha_{n-1})\overline{V}^{n-1}(R^n) + \alpha_{n-1}\hat{v}^n & \text{if } y = R^n, \\ \overline{V}^n(y) \vee \left\{(1-\alpha_{n-1})\overline{V}^{n-1}(R^n) + \alpha_{n-1}\hat{v}^n\right\} & \text{if } y > R^n, \\ \overline{V}^n(y) \wedge \left\{(1-\alpha_{n-1})\overline{V}^{n-1}(R^n) + \alpha_{n-1}\hat{v}^n\right\} & \text{if } y < R^n, \end{cases} \tag{19.1}$$

where $x \wedge y = \max\{x, y\}$, and $x \vee y = \min\{x, y\}$. Equation (19.1) starts by updating the slope $\overline{V}^n(y)$ for $y = R^n$. We then want to make sure that the slopes are declining. So, if we find a slope to the right that is larger, we simply bring it down to our estimated slope for $y = R^n$. Similarly, if there is a slope to the left that is smaller, we simply raise it to the slope for $y = R^n$. The steps are illustrated in figure 19.1.

The leveling algorithm is easy to visualize, but it is unlikely to be the best way to maintain monotonicity. For example, we may update a value at $y = R^n$ for which there are very few observations. But because it produces an unusually high or low estimate, we find ourselves simply forcing other slopes higher or lower just to maintain monotonicity.

19.1.0.2 The SPAR algorithm A more elegant strategy is the SPAR (separable projective approximation routine) which works as follows. Assume that we start with our original set of values $(\overline{V}^{n-1}(y))_{y \geq 0}$, and that we sample $y = R^n$ and obtain an estimate of the slope $\hat{v}^n$. After the update, we obtain the set of values (which we store temporarily in the function $\bar{y}^n(y)$)

$$\bar{z}^n(y) = \begin{cases} (1-\alpha_{n-1})\overline{V}^{n-1}(y) + \alpha_{n-1}\hat{v}^n, & y = R^n \\ \overline{V}^{n-1}(y) & \text{otherwise.} \end{cases} \tag{19.2}$$

If $\bar{z}^n(y) \geq \bar{z}^n(y+1)$ for all y, then we are in good shape. If not, then either $\bar{z}^n(R^n) < \bar{z}^n(R^n+1)$ or $\bar{z}^n(R^n-1) < \bar{z}^n(R^n)$. We can fix the problem by solving the projection problem

$$\min_v \|v - \bar{z}^n\|^2. \tag{19.3}$$

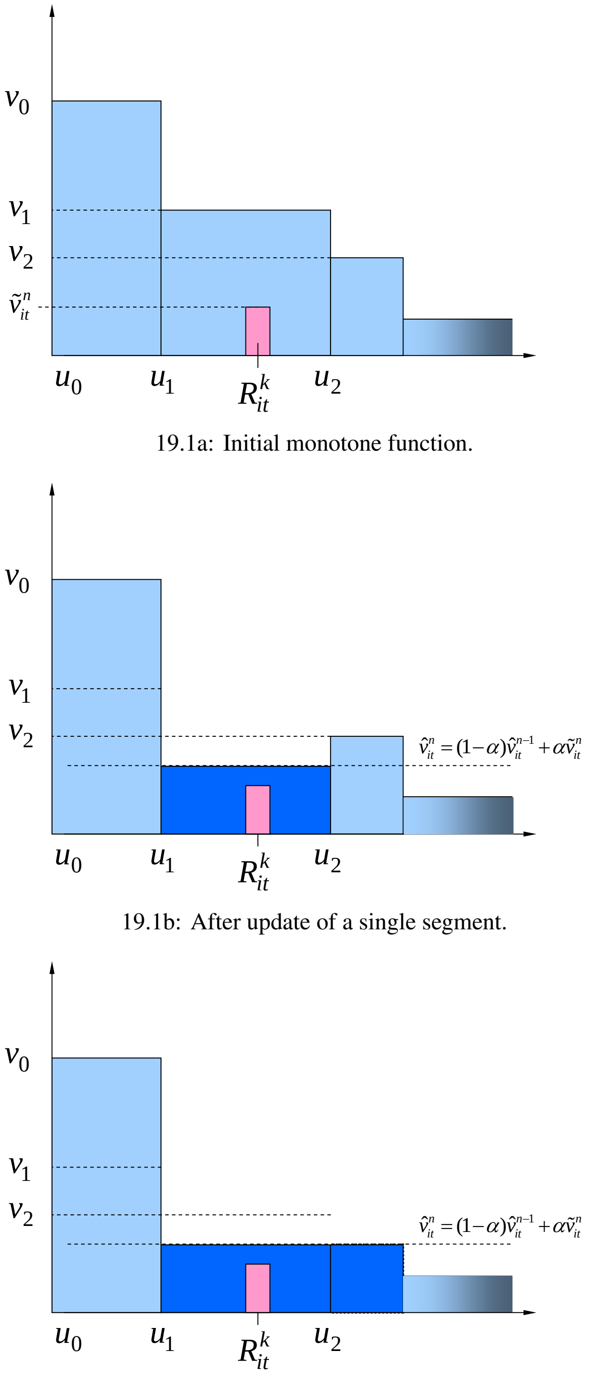
subject to

$$v(z+1) - v(z) \quad \leq \quad 0. \tag{19.4}$$

Solving this projection is especially easy. Imagine that after our update, we have a violation to the left. The projection is achieved by averaging the updated cell with all the cells to the left that create a monotonicity violation. This means that we want to find the largest $i \leq R^n$ such that

$$\bar{z}^n(i-1) \geq \frac{1}{R^n - i + 1} \sum_{y=i}^{R^n} \bar{z}^n(y).$$

In other words, we can start by averaging the values for R^n and $R^n - 1$ and checking to see if we now have a concave function. If not, we keep lowering the left end of the range until we either restore monotonicity or reach $y = 0$. If our monotonicity violation is to the right, then we repeat the process to the right.


v_0
v_1
v_2
$\tilde{v}^n_{it}$
u_0
u_1
R^k_{it}
u_2
$\hat{v}^n_{it} = (1-\alpha)\hat{v}^{n-1}_{it} + \alpha\tilde{v}^n_{it}$

19.1a: Initial monotone function.

19.1b: After update of a single segment.

19.1c: After leveling operation.

Figure 19.1 Steps of the leveling algorithm. Figure 19.1a shows the initial monotone function, with the observed R and observed value of the function $\hat{v}$. Figure 19.1b shows the function after updating the single segment, producing a non-monotone function. Figure 19.1c shows the function after monotonicity restored by leveling the function.

The steps of the algorithm are given in figure 19.2, with an illustration given in figure 19.3. We start with a monotone set of values (a), then update one of the values to produce a monotonicity violation (b), and finally average the violating values together to restore monotonicity (c).

There are a number of variations of these algorithms that help with convergence. For example, in the SPAR algorithm, we can solve a weighted projection that gives more weight to slopes that have received more observations. To do this, we weight each value of $\bar{y}(r)$ by the number of observations that segment r has received when computing $\bar{y}^n(i-1)$.

19.1.0.3 The CAVE algorithm A particularly useful variation is to perform an initial update (when we compute $\bar{y}$) over a wider interval than just $y = R^n$. Assume we are given a parameter δ^0 which has been chosen so that it is approximately 20 to 50 percent of the maximum value that R^n might take. Now compute $\bar{z}(y)$ using

$$\bar{z}^n(y) = \begin{cases} (1-\alpha_{n-1})\overline{V}^{n-1}(y) + \alpha_{n-1}\hat{v}^n, & R^n - \delta^n \leq y \leq R^n + \delta^n, \\ \overline{V}^{n-1}(y) & \text{otherwise.} \end{cases}$$

Here, we are using $\hat{v}^n$ to update a wider range of the interval. We then apply the same logic for maintaining monotonicity (concavity if these are slopes). We start with the interval $R^n \pm \delta^0$, but we have to periodically reduce δ^0. We might, for example, track the objective function (call it F^n), and update the range using

$$\delta^n = \begin{cases} \delta^{n-1} & \text{If } F^n \geq F^{n-1} - \epsilon, \\ \max\{1, .5\delta^{n-1}\} & \text{otherwise.} \end{cases}$$

While the rules for reducing δ^n are generally ad hoc, we have found that this is critical for fast convergence. The key is that we have to pick δ^0 so that it plays a critical scaling role, since it has to be set so that it is roughly on the order of the maximum value that R^n can take. If SPAR or the leveling algorithm are going to be successful, then these will have to be adapted to solve these scaling problems.

Step 0 Initialize $\overline{V}^0$ and set $n = 1$.

Step 1 Sample R^n.

Step 2 Observe a sample of the value function $\hat{v}^n$.

Step 3 Calculate the vector z^n as follows

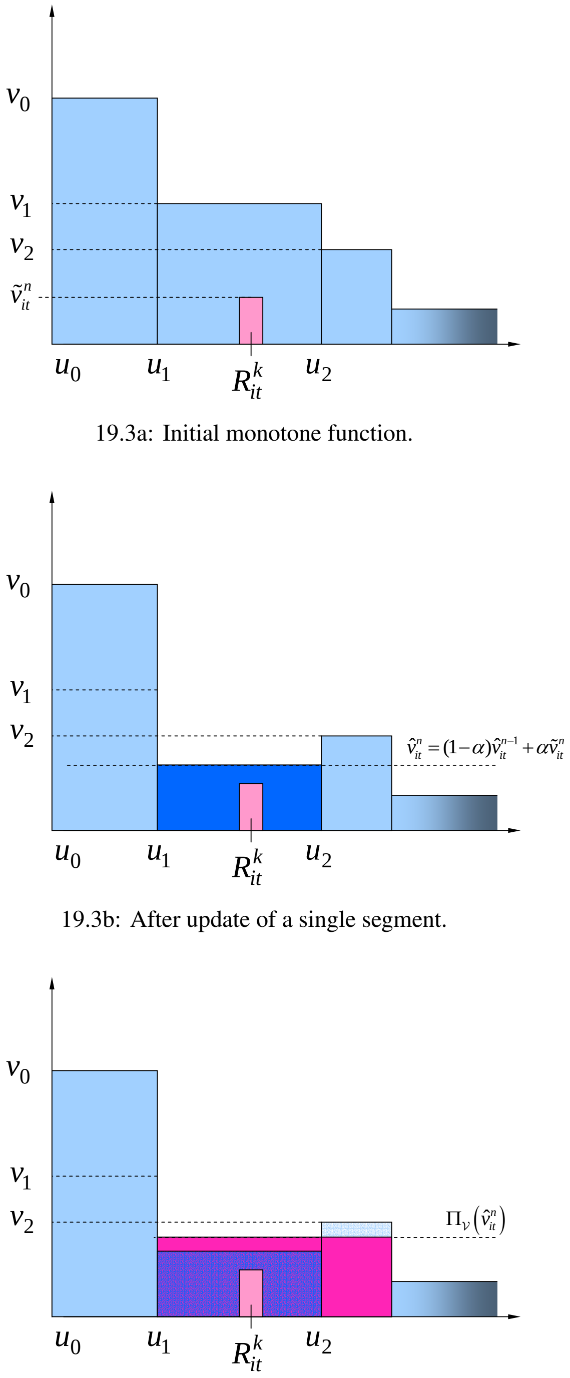
$$z^n(y) = \begin{cases} (1-\alpha_{n-1})V_{R^n}^{n-1} + \alpha_{n-1}\hat{v}^n & \text{if } y = R^n, \\ v^{n-1}(y) & \text{otherwise} \end{cases}$$

Step 4 Project the updated estimate onto the space of monotone functions:

$$v^n = \Pi(z^n),$$

by solving (19.3)-(19.4). Increase n by one and go to Step 1.

Figure 19.2 The learning form of the separable, projective approximation routine (SPAR).


v_0
v_1
v_2
$\tilde{v}_{it}^n$
u_0
u_1
R_{it}^k
u_2
$\hat{v}_{it}^n = (1-\alpha)\hat{v}_{it}^{n-1} + \alpha\tilde{v}_{it}^n$
$\Pi_{\mathcal{V}}\left(\hat{v}_{it}^n\right)$

19.3a: Initial monotone function.

19.3b: After update of a single segment.

19.3c: After projection.

Figure 19.3 Steps of the SPAR algorithm. Figure 19.3a shows the initial monotone function, with the observed R and observed value of the function $\hat{v}$. Figure 19.3b shows the function after updating the single segment, producing a non-monotone function. Figure 19.3c shows the function after the projection operation.

19.2 MULTIPLE CONVEX DIMENSIONS

19.2.1 Benders decomposition

19.2.1.1 SDDP - Intertemporal independence

19.2.1.2 Regularization

19.2.2 Benders with exogenous state variable

19.3 HIGH-DIMENSIONAL APPLICATIONS

Multiattribute
Linear VFA
Hierarchical

In chapter 3, we introduced general purpose approximation tools for approximating functions without assuming any special structural properties. In this chapter, we focus on approximating value functions that arise in resource allocation problems. For example, if R is the amount of resource available (water, oil, money, vaccines) and $V(R)$ is the value of having R units of our resource, we often find that $V(R)$ might be linear (or approximately linear), nonlinear (concave), piecewise linear, or in some cases, simply something continuous. Value functions with this structure yield to special approximation strategies.

We consider a series of strategies for approximating the value function using increasing sophistication:

Linear approximations - These are typically the simplest nontrivial approximations that work well when the functions are approximately linear over the range of interest. It is important to realize that we mean "linear in the state" as opposed to the more classical "linear in the parameters" model that we considered earlier.

Separable, piecewise linear, concave (convex if minimizing) - These functions are especially useful when we are interested in integer solutions. Separable functions are relatively easy to estimate and offer special structural properties when solving the optimality equations.

Auxiliary functions - This is a special class of algorithms that fixes an initial approximation and uses stochastic gradients to adjust the function.

General nonlinear regression equations - Here, we bring the full range of tools available from the field of regression.

Cutting planes - This is a technique for approximating multidimensional, piecewise linear functions that has proven to be particularly powerful for multistage linear programs such those that arise in dynamic resource allocation problems.

An important dimension of this chapter will be our use of derivatives to estimate value functions, rather than just the value of being in a state. When we want to determine how much oil should be sent to a storage facility, what matters most is the marginal value of additional oil. For some problem classes, this is a particularly powerful device that dramatically improves convergence.

19.4 VALUES VERSUS MARGINAL VALUES

It is common in dynamic programming to talk about the problem of estimating the value of being in a state. There are many applications where it is more useful to work with the derivative or gradient of the value function. In one community, where "heuristic dynamic programming" represents approximate dynamic programming based on estimating the value of being in a state, "dual heuristic programming" refers to approximating the gradient.

We are going to use the context of resource allocation problems to illustrate the power of using the gradient. In principal, the challenge of estimating the slope of a function is the same as that of estimating the function itself (the slope is simply a different function). However, there can be important, practical advantages to estimating slopes. First, if the function is approximately linear, it may be possible to replace estimates of the value of being in each state (or set of states) with a single parameter which is the estimate of the slope of the function. Estimating constant terms is typically unnecessary.

A second and equally important difference is that if we estimate the value of being in a state, we get one estimate of the value of being in a state when we visit that state. When we estimate a gradient, we get an estimate of a derivative for each type of resource. For example, if $R_t = (R_{tr})_{r \in \mathcal{R}}$ is our resource vector and $V_t(R_t)$ is our value function, then the gradient of the value function with respect to R_t would look like

$$\nabla_{R_t} V_t(R_t) = \begin{pmatrix} \hat{v}_{tr_1} \\ \hat{v}_{tr_2} \\ \vdots \\ \hat{v}_{tr_{|\mathcal{A}|}} \end{pmatrix},$$

where

$$\hat{v}_{tr_i} = \frac{\partial V_t(R_t)}{\partial R_{tr_i}}.$$

There may be additional work required to obtain each element of the gradient, but the incremental work can be far less than the work required to get the value function itself. This is particularly true when the optimization problem naturally returns these gradients (for example, dual variables from a linear program), but this can even be true when we have to resort to numerical derivatives. Once we have all the calculations to solve a problem once, solving small perturbations can be relatively inexpensive.

There is one important problem class where finding the value of being in a state is equivalent to finding the derivative. That is the case of managing a single resource. In this case, the state of our system (the resource) is the attribute vector r, and we are interested in estimating the value $V(r)$ of our resource being in state r. Alternatively, we can represent the state of our system using the vector R_t, where $R_{tr} = 1$ indicates that our resource has attribute r (we assume that $\sum_{r \in \mathcal{R}} R_{tr} = 1$). In this case, the value function can be written

$$V_t(R_t) = \sum_{r \in \mathcal{R}} v_{tr} R_{tr}.$$

Here, the coefficient v_{tr} is the derivative of $V_t(R_t)$ with respect to R_{tr}.

In a typical implementation of an approximate dynamic programming algorithm, we would only estimate the value of a resource when it is in a particular state (given by the vector r). This is equivalent to finding the derivative $\hat{v}_r$ only for the value of r where

$R_{tr} = 1$. By contrast, computing the gradient $\nabla_{R_t} V_t(R_t)$ implicitly assumes that we are computing $\hat{v}_r$ for each $r \in \mathcal{R}$. There are some algorithmic strategies (we will describe an example of this in section 19.9) where this assumption is implicit in the algorithm. Computing $\hat{v}_r$ for all $r \in \mathcal{R}$ is reasonable if the attribute state space is not too large (for example, if r is a physical location among a set of several hundred locations). If r is a vector, then enumerating the attribute space can be prohibitive (it is, in effect, the "curse of dimensionality" revisited).

Given these issues, it is critical to first determine whether it is necessary to estimate the slope of the value function, or the value function itself. The result can have a significant impact on the algorithmic strategy.

19.5 LINEAR APPROXIMATIONS

There are a number of problems where we are allocating resources of different types. As in the past, we let r be the attributes of a resource and R_{tr} be the quantity of resources with attribute r in our system at time t with $R_t = (R_{tr})_{r \in \mathcal{R}}$. R_t may describe our investments in different resource classes (growth stocks, value stocks, index funds, international mutual funds, domestic stock funds, bond funds). Or R_t might be the amount of oil we have in different reserves or the number of people in a management consulting firm with particular skill sets. We want to make decisions to acquire or sell resources of each type, and we want to capture the impact of decisions now on the future through a value function $V_t(R_t)$.

Rather than attempt to estimate $V_t(R_t)$ for each value of R_t, it may make more sense to estimate a linear approximation of the value function with respect to the resource vector. Linear approximations can work well when the single-period contribution function is continuous and increases or decreases monotonically over the range we are interested in (the function may or may not be differentiable). They can also work well in settings where the value function increases or decreases monotonically, even if the value function is neither convex nor concave, nor even continuous.

To illustrate, consider the problem of purchasing a commodity. Let

$\hat{D}_t$ = The random demand during time interval t,

R_t = The resources on hand at time t just before we make an ordering decision,

x_t = The quantity ordered at time t to be used during time interval $t+1$,

R_t^x = The resources available just after we make a decision,

$\hat{p}_t$ = The market price for selling commodities during time interval t,

c_t = The purchase cost for commodities purchased at time t.

At time t, we know the price $\hat{p}_t$ and demand $\hat{D}_t$ for time interval t, but we have to choose how much to order for the next time interval. The transition equations are given by

$$\begin{aligned} R_t^x &= R_t + x_t, \\ R_{t+1} &= [R_t^x - \hat{D}_{t+1}]^+. \end{aligned}$$

The value of being in state R_t is given by

$$V_t(R_t) = \max_{x_t} \mathbb{E}\big(\hat{p}_{t+1} \min\{R_t + x_t, \hat{D}_{t+1}\} - c_t x_t + V_t^x(R_t + x_t)\big), \tag{19.5}$$

where $V_t^x(R_t^x)$ is the post-decision value function, while $V_t(R_t)$ is the traditional value function around the pre-decision state. Now assume that we introduce a linear value function approximation

$$\overline{V}_t^x(R_t^x) \approx \bar{v}_t R_t^x.$$

The resulting approximation can be written

$$\begin{aligned}\widetilde{V}_t(R_t) &= \max_{x_t} \mathbb{E}\big(\hat{p}_{t+1} \min\{R_t + x_t, \hat{D}_{t+1}\} - c_t x_t + \bar{v}_t R_t^x\big) \\ &= \max_{x_t} \mathbb{E}\big(\hat{p}_{t+1} \min\{R_t + x_t, \hat{D}_{t+1}\} - c_t x_t + \bar{v}_t (R_t + x_t)\big). \quad (19.6)\end{aligned}$$

We assume that we can compute, or at least approximate, the expectation in equation (19.6). If this is the case, we may approximate the gradient at iteration n using a numerical derivative, as in

$$\hat{v}_t = \widetilde{V}_t(R_t + 1) - \widetilde{V}_t(R_t).$$

We now use $\hat{v}_t$ to update the value function $\overline{V}_{t-1}$ using

$$\bar{v}_{t-1} \leftarrow (1-\alpha)\bar{v}_{t-1} + \alpha \hat{v}_t.$$

Normally, we would use $\hat{v}_t$ to update $\overline{V}_{t-1}(R_{t-1}^x)$ around the previous post-decision state variable R_{t-1}^x. Linear approximations, however, are a special case, since the slope is the same for all R_{t-1}^x, which means it is also the same for $R_{t-1} = R_{t-1}^x - x_{t-1}$.

Linear approximations are useful in two settings. First, the value function may be approximately linear over the range that we are interested in. Imagine, for example, that you are trying to decide how many shares of stock you want to sell, where the range is between 0 and 1,000. As an individual investor, it is unlikely that selling all 1,000 shares will change the market price. However, if you are a large mutual fund and you are trying to decide how many of your 50 million shares you want to sell, it is quite likely that such a high volume would, in fact, move the market price. When this happens, we need a nonlinear function.

A second use of linear approximations arises when managing resources such as people and complex equipment such as locomotives or aircraft. Let r be the attributes of a resource and R_{tr} be the number of resources with attribute r at time t. Then it is likely that R_{tr} will be 0 or 1, implying that a linear function is all we need. For these problems, a linear value function is particularly convenient because it means we need one parameter, $\bar{v}_{tr}$, for each attribute r.

19.6 PIECEWISE LINEAR FUNCTIONS FOR RESOURCE ALLOCATION

Scalar, piecewise linear functions have proven to be an exceptionally powerful way of solving high dimensional stochastic resource allocation problems. We can describe the algorithm with a minimum of technical details using what is known as a “plant-warehouse-customer” model, which is a form of multidimensional newsvendor problem. Imagine that we have the problem depicted in figure 19.4a. We start by shipping “product” out of the four “plant” nodes on the left, and we have to decide how much to send to each of the five “warehouse” nodes in the middle. After making this decision, we then observe the demands at the five “customer” nodes on the right.

We can solve this problem using separable, piecewise linear value function approximations. Assume we have an initial estimate of a piecewise linear value function for resources at the warehouses (setting these equal to zero is fine). This gives us the network shown in figure 19.4b, which is a small linear program (even when we have hundreds of plant and warehouse nodes). Solving this problem gives us a solution of how much to send to each node.

We then use the solution to the first stage (which gives us the resources available at each warehouse node), take a Monte Carlo sample of each of the demands, and solve a second linear program that sends product from each warehouse to each customer. What we want from this stage is the dual variable for each warehouse node, which gives us an estimate of the marginal value of resources at each node. Note that some care needs to be used here, because these dual variables are not actually estimates of the value of one more resources, but rather are subgradients, which means that they may be the value of the last resource or the next resource, or something in between.

Finally, we use these dual variables to update the piecewise linear value functions using the methods described above. This process is repeated until the solution no longer seems to be improving.

Although we have described this algorithm in the context of a two-stage problem, the same basic strategy can be applied for problems with many time periods. Using approximate value iteration (TD(0)), we would step forward in time, and after solving each linear program we would stop and use the duals to update the value functions from the previous time period (more specifically, around the previous post-decision state). For a finite horizon problem, we would proceed until the last time period, then repeat the entire process until the solution seems to be converging.

With more work, we can implement a backward pass (TD(1)) by avoiding any value function updates until we reach the final time period, but we would have to retain information about the effect of incrementing the resources at each node by one unit (this is best done with a numerical derivative). We would then need to step back in time, computing the marginal value of one more resource at time t using information about the value of one more resource at time $t + 1$. These marginal values would be used to update the value function approximations.

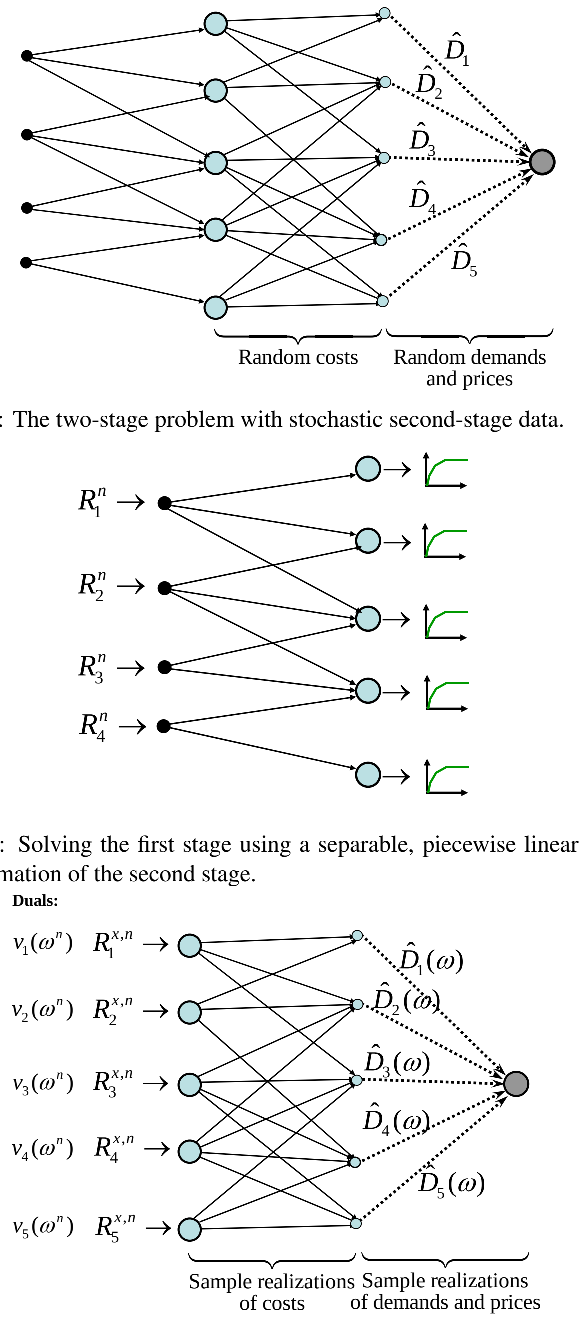
This algorithmic strategy has some nice features:

- This is a very general model with applications that span equipment, people, product, money, energy and vaccines. It is ideally suited for "single layer" resource allocation problems (one type of resource, rather than pairs such as pilots and aircraft, locomotives and trains or doctors and patients), although many two-layer problems can be reasonably approximated as single-layer problems.

- The methodology scales to very large problems, with hundreds or thousands of nodes, and tens of thousands of dimensions in the decision vector.

- We do not need to solve the exploration-exploitation problem. A pure exploitation strategy works fine. The reason has to do with the concavity of the value function approximations, which has the effect of pushing suboptimal value functions toward the correct solution.

- Piecewise linear value function approximations are quite robust, and avoid making any simplifying assumptions about the shapes of the value functions.


$\hat{D}_1$
$\hat{D}_2$
$\hat{D}_3$
$\hat{D}_4$
$\hat{D}_5$
Random costs
Random demands and prices
R_1^n
R_2^n
R_3^n
R_4^n
Duals:
$v_1(\omega^n)$ $R_1^{x,n}$
$v_2(\omega^n)$ $R_2^{x,n}$
$v_3(\omega^n)$ $R_3^{x,n}$
$v_4(\omega^n)$ $R_4^{x,n}$
$v_5(\omega^n)$ $R_5^{x,n}$
$\hat{D}_1(\omega)$
$\hat{D}_2(\omega)$
$\hat{D}_3(\omega)$
$\hat{D}_4(\omega)$
$\hat{D}_5(\omega)$
Sample realizations of costs
Sample realizations of demands and prices

19.4a: The two-stage problem with stochastic second-stage data.

19.4b: Solving the first stage using a separable, piecewise linear approximation of the second stage.

19.4c: Solving a Monte Carlo realization of the second stage and obtaining dual variables.

Figure 19.4 Steps in estimating separable, piecewise-linear approximations for two-stage stochastic programs.

19.7 REGRESSION METHODS

As in chapter 3 we can create regression models where the basis functions are manipulations of the number of resources of each type. For example, we might use

$$\overline{V}(R) = \theta_0 + \sum_{a \in \mathcal{A}} \theta_{1a} R_a + \sum_{a \in \mathcal{A}} \theta_{2a} R_a^2, \tag{19.7}$$

where $\theta = (\theta_0, (\theta_{1r})_{r \in \mathcal{R}}, (\theta_{2r})_{r \in \mathcal{R}})$ is a vector of parameters that are to be determined. The choice of explanatory terms in our approximation will generally reflect an understanding of the properties of our problem. For example, equation (19.7) assumes that we can use a mixture of linear and separable quadratic terms. A more general representation is to assume that we have developed a family $\mathcal{F}$ of basis functions $(\phi_f(R))_{f \in \mathcal{F}}$. Examples of a basis function are

$$\begin{aligned}
\phi_f(R) &= R_{r_f}^2, \\
\phi_f(R) &= \left(\sum_{r \in \mathcal{R}_f} R_r \right)^2 \quad \text{for some subset } \mathcal{R}_f, \\
\phi_f(R) &= (R_{r_1} - R_{r_2})^2, \\
\phi_f(R) &= |R_{r_1} - R_{r_2}|.
\end{aligned}$$

A common strategy is to capture the number of resources at some level of aggregation. For example, if we are purchasing emergency equipment, we may care about how many pieces we have in each region of the country, and we may also care about how many pieces of a type of equipment we have (regardless of location). These issues can be captured using a family of aggregation functions G_f, $f \in \mathcal{F}$, where $G_f(r)$ aggregates an attribute vector r into a space $\mathcal{R}^{(f)}$ where for every basis function f there is an element $r_f \in \mathcal{R}^{(f)}$. Our basis function might then be expressed using

$$\phi_f(R) = \sum_{r \in \mathcal{R}} 1_{\{G_f(r) = r_f\}} R_r.$$

As we originally introduced in section 18.4.1, the explanatory variables used in the examples above, which are generally referred to as independent variables in the regression literature, are typically referred to as basis functions by the approximate dynamic programming community. A basis function can be linear, nonlinear separable, nonlinear nonseparable, and even nondifferentiable, although the nondifferentiable case will introduce additional technical issues. The challenge, of course, is that it is the responsibility of the modeler to devise these functions for each application. We have written our basis functions purely in terms of the resource vector, but it is possible for them to be written in terms of other parameters in a more complex state vector, such as asset prices.

Given a set of basis functions, we can write our value function approximation as

$$\overline{V}(R|\theta) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(R). \tag{19.8}$$

It is important to keep in mind that $\overline{V}(R|\theta)$ (or more generally, $\overline{V}(S|\theta)$), is any functional form that approximates the value function as a function of the state vector parameterized by θ. Equation (19.8) is a classic linear-in-the-parameters function. We are not constrained to this form, but it is the simplest and offers some algorithmic shortcuts.

The issues that we encounter in formulating and estimating $\overline{V}(R|\theta)$ are the same that any student of statistical regression would face when modeling a complex problem. The major difference is that our data arrives over time (iterations), and we have to update our formulas recursively. Also, it is typically the case that our observations are nonstationary. This is particularly true when an update of a value function depends on an approximation of the value function in the future (as occurs with value iteration or any of the TD(λ) classes of algorithms). When we are estimating parameters from nonstationary data, we do not want to equally weight all observations.

The problem of finding θ can be posed in terms of solving the following stochastic optimization problem

$$\min_\theta \mathbb{E}\frac{1}{2}(\overline{V}(R|\theta) - \hat{V})^2.$$

We can solve this using a stochastic gradient algorithm, which produces updates of the form

$$\begin{aligned}\bar{\theta}^n &= \bar{\theta}^{n-1} - \alpha_{n-1}(\overline{V}(R^n|\bar{\theta}^{n-1}) - \hat{V}(\omega^n))\nabla_\theta \overline{V}(R^n|\theta^n)\\ &= \bar{\theta}^{n-1} - \alpha_{n-1}(\overline{V}(R^n|\bar{\theta}^{n-1}) - \hat{V}(\omega^n))\begin{pmatrix}\phi_1(R^n)\\ \phi_2(R^n)\\ \vdots\\ \phi_F(R^n)\end{pmatrix}.\end{aligned}$$

If our value function is linear in R_t, we would write

$$\overline{V}(R|\theta) = \sum_{r\in\mathcal{R}} \theta_r R_r.$$

In this case, our number of parameters has shrunk from the number of possible realizations of the entire vector R_t to the size of the attribute space (which, for some problems, can still be large, but nowhere near as large as the original state space). For this problem, $\phi(R^n) = R^n$.

It is not necessarily the case that we will always want to use a linear-in-the-parameters model. We may consider a model where the value increases with the number of resources, but at a declining rate that we do not know. Such a model could be captured with the representation

$$\overline{V}(R|\theta) = \sum_{r\in\mathcal{R}} \theta_{1r} R_r^{\theta_{2r}},$$

where we expect $\theta_2 < 1$ to produce a concave function. Now, our updating formula will look like

$$\begin{aligned}\theta_1^n &= \theta_1^{n-1} - \alpha_{n-1}(\overline{V}(R^n|\bar{\theta}^{n-1}) - \hat{V}(\omega^n))(R^n)^{\theta_2},\\ \theta_2^n &= \theta_2^{n-1} - \alpha_{n-1}(\overline{V}(R^n|\bar{\theta}^{n-1}) - \hat{V}(\omega^n))(R^n)^{\theta_2}\ln R^n\end{aligned}$$

where we assume the exponentiation operator in $(R^n)^{\theta_2}$ is performed componentwise.

We can put this updating strategy in terms of temporal differencing. As before, the temporal difference is given by

$$\delta_\tau = C_\tau(R_\tau, x_{\tau+1}) + \overline{V}_{\tau+1}^{n-1}(R_{\tau+1}) - \overline{V}_\tau^{n-1}(R_\tau).$$

The original parameter updating formula (equation (17.7)) when we had one parameter per state now becomes

$$\bar{\theta}^n = \bar{\theta}_t^{n-1} + \alpha_{n-1} \sum_{\tau=t}^{T} \lambda^{\tau-t} \delta_\tau \nabla_\theta \overline{V}(R^n|\bar{\theta}^n).$$

It is important to note that in contrast with most of our other applications of stochastic gradients, updating the parameter vector using gradients of the objective function requires mixing the units of θ with the units of the value function. In these applications, the stepsize α_{n-1} has to also perform a scaling role.

19.8 VALUE ITERATION FOR MULTIDIMENSIONAL DECISION VECTORS

In the previous section we saw that the use of the post-decision state variable meant that the process of choosing the best action required solving a deterministic optimization problem. This opens the door to considering problems where the decision is a vector x_t. For example, imagine that we have a problem of assigning an agent in a multiskill call center to customers requiring help with their computers. An agent of type i might have a particular set of language and technical skills. A customer has answered a series of automated questions, which we capture with a label j. Let

R_{ti} = The number of agents available at time t with skill set i,

D_{tj} = The number of customers waiting at time t whose queries are characterized by j.

We let $R_t = (R_{ti})_i$ and $D_t = (D_{tj})_j$, and finally let our state variable be $S_t = (R_t, D_t)$. Let our decision vector be defined using

x_{tij} = Number of agents of type i who are assigned to customers of type j at time t,

x_t = $(x_{tij})_{i,j}$.

For realistic problems, it is easy to create vectors x_t with hundreds or thousands of dimensions. Finally, let

c_{ij} = Estimated time required for an agent of type i to serve a customer of type j.

The state variables evolve according to

$$\begin{aligned} R_{t+1,i} &= R_{ti} - \sum_j x_{tij} + \hat{R}_{t+1,i}, \\ D_{t+1,j} &= D_{tj} - \sum_i x_{tij} + \hat{D}_{t+1,j}. \end{aligned}$$

Here, $\hat{R}_{t+1,i}$ represents the number of agents that were busy but which became idle between t and $t+1$ because they completed their previous assignment. $\hat{D}_{t+1,j}$ represents arrival of new customers. We note that R_t and D_t are pre-decision state variables. Their post-decision counterparts are given by

$$R^x_{t+1,i} = R_{ti} - \sum_j x_{tij}, \tag{19.9}$$

$$D^x_{t+1,j} = D_{tj} - \sum_i x_{tij}. \tag{19.10}$$

We are going to have to create a value function approximation $\overline{V}(S^x_t)$. For the purpose of illustrating the basic idea, assume we use a separable approximation, which we can write using

$$\overline{V}_t(R^x_t, D^x_t) = \sum_i \overline{V}^R_{ti}(R^x_{ti}) + \sum_j \overline{V}^D_{tj}(D^x_{tj}).$$

Since we are minimizing, we intend to create scalar, convex approximations for $\overline{V}^R_{ti}(R^x_{ti})$ and $\overline{V}^D_{tj}(D^x_{tj})$. Now, our VFA policy requires solving a linear (or nonlinear) math programming problem of the form

$$\min_{x_t} \sum_i \sum_j c_{ij} x_{tij} + \sum_i \overline{V}^R_{t-1,i}(R^x_{ti}) + \sum_j \overline{V}^D_{t-1,j}(D^x_{tj}) \tag{19.11}$$

where R^x_{ti} and D^x_{tj} are given by (19.9) and (19.10), respectively. Also, our optimization problem has to be solved subject to the constraints

$$\sum_j x_{tij} \leq R_{ti}, \tag{19.12}$$

$$\sum_i x_{tij} \leq D_{tj}, \tag{19.13}$$

$$x_{tij} \geq 0. \tag{19.14}$$

The optimization problem described by equations (19.11) - (19.14) is a linear or nonlinear optimization problem. If the scalar, separable value function approximations are convex, this is generally fairly easy to solve, even when x_t has hundreds or thousands of dimensions.

Elsewhere, we would let $\hat{v}^n_t$ be the value of the optimal objective function, which we then use to update the value function approximation. For this problem class, we are going to take advantage of the fact that the optimal solution will yield dual variables for the constraints (19.12) and (19.13). Call these dual variables $\hat{v}^R_{ti}$ and $\hat{v}^D_{tj}$, respectively. Thus, we can interpret $\hat{v}^R_{ti}$ as an approximation of the marginal value of R_{ti}, while $\hat{v}^D_{tj}$ is an approximation of the marginal value of D_{tj}. We can use these marginal values to update our value function approximations. However, we would use $\hat{v}^R_t$ and $\hat{v}^D_t$ to update the value function approximations $\overline{V}^R_{t-1}(R^x_{t-1})$ and $\overline{V}^D_{t-1}(D^x_{t-1})$, which means we are using information from the problem we solve at time t to update value function approximations at time $t-1$ around the previous, post-decision state variable.

19.9 CUTTING PLANES FOR MULTIDIMENSIONAL FUNCTIONS

Cutting planes represent a powerful strategy for representing concave (or convex if we are minimizing), piecewise-linear functions for multidimensional problems. This method evolved originally not as a method for approximating dynamic programs, but instead as a technique for solving linear programs in the presence of uncertainty. In the 1950's, the research community recognized that many optimization problems involve different forms of uncertainty, with the most common being the challenge of allocating resources now to serve demands in the future that have not yet been realized. For this reason, a subcommunity within math programming, known as the stochastic programming community, has developed a rich theory and some powerful algorithmic strategies for handling uncertainty within linear programs and, more recently, integer programs.

Historically, dynamic programming has been viewed as a technique for small, discrete optimization problems, while stochastic programming has been the field that handles uncertainty within math programs (which are typically characterized by high-dimensional decision vectors and large numbers of constraints). The connections between stochastic programming and dynamic programming, historically viewed as diametrically competing frameworks, have been largely overlooked. This section is designed to bridge the gap between stochastic programming and approximate dynamic programming. Our presentation is facilitated by notational decisions (in particular the use of x as our decision vector) that we made in the beginning of the book.

19.9.1 Convexity with exogenous information state

Information state is purely exogenous
Information state influenced by decisions

19.10 WHY DOES IT WORK?**

19.10.1 The projection operation

Let v_s^{n-1} be the value (or the marginal value) of being in state s at iteration $n-1$ and assume that we have a function where we know that we should have $v_{s+1}^{n-1} \geq v_s^{n-1}$ (we refer to this function as *monotone*). For example, if this is marginal values, we would expect this if we were describing a concave function. Now assume that we have a sample realization $\hat{v}_s^n$ which is the value (or marginal value) of being in state s. We would then smooth this new observation with the previous estimates using

$$z_s^n = \begin{cases} (1-\alpha_{n-1})v_s^{n-1} + \alpha_{n-1}\hat{v}^n & \text{if } s = s^n, \\ v_s^n & \text{otherwise.} \end{cases} \tag{19.15}$$

Since $\hat{v}_s^n$ is random, we cannot expect z_s^n to also be monotone. In this section, we want to restore monotonicity by defining an operator $v^n = \Pi_V(z)$ where $v_{s+1}^n \geq v_s^n$. There are several ways to do this. In this section, we define the operator $v = \Pi_V(z)$, which takes a vector z (which is not necessarily monotone) and produces a monotone vector v. If we wish to find v that is as close as possible to z, we would solve

$$\begin{aligned} &\min \frac{1}{2}\|v - z\|^2 \\ &\text{subject to: } v_{s+1} - v_s \leq 0, \quad s = 0, \ldots, M. \end{aligned} \tag{19.16}$$

Assume that v_0 is bounded above by B, and v_{M+1}, for $s < M$, is bounded from below by $-B$. Let $\lambda_s \geq 0$, $s = 0, 1, \ldots, M$ be the Lagrange multipliers associated with equation

(19.16). It is easy to see that the optimality equations are

$$\begin{aligned} v_s &= z_s + \lambda_s - \lambda_{s-1}, \quad s = 1, 2, \ldots, M, & (19.17) \\ \lambda_s(v_{s+1} - v_s) &= 0, \quad s = 0, 1, \ldots, M. & (19.18) \end{aligned}$$

Let $i_1, \ldots, i_2$ be a sequence of states where

$$v_{i_1-1} > v_{i_1} = v_{i_1+1} = \cdots = c = \cdots = v_{i_2-1} = v_{i_2} > v_{i_2+1}.$$

We can then add equations (19.17) from i_1 to i_2 to yield

$$c = \frac{1}{i_2 - i_1 + 1} \sum_{s=i_1}^{i_2} z_s.$$

If $i_1 = 1$, then c is the smaller of the above and B. Similarly, if $i_2 = M$, then c is the larger of the above and $-B$.

We also note that $v^{n-1} \in \mathcal{V}$ and z^n computed by (19.15) differs from v^{n-1} in just one coordinate. If $z^n \notin \mathcal{V}$ then either $z^n_{s^n-1} < z^n_{s^n}$, or $z^n_{s^n+1} > z^n_{s^n}$.

If $z^n_{s^n-1} < z^n_{s^n}$, then we we need to find the largest $1 < i \leq s^n$ where

$$z^n_{i-1} \geq \frac{1}{s^n - i + 1} \sum_{s=i}^{s^n} z^n_s.$$

If i cannot be found, then we use $i = 1$. We then compute

$$c = \frac{1}{s^n - i + 1} \sum_{s=i}^{s^n} z^n_s$$

and let

$$v^{n+1}_j = \min(B, c), \quad j = i, \ldots, s^n.$$

We have $\lambda_0 = \max(0, c - B)$, and

$$\lambda_s = \begin{cases} 0 & s = 1, \ldots, i-1, \\ \lambda_{s-1} + z_s - v_s & s = i, \ldots, s^n - 1, \\ 0 & s = s^n, \ldots, M. \end{cases}$$

It is easy to show that the solution found and the Lagrange multipliers satisfy equations (19.17) -(19.18).

If $z^n_{s^n} < z^n_{s^n+1}$, then the entire procedure is basically the same with appropriate inequalities reversed.

19.11 BIBLIOGRAPHIC NOTES

Section 19.4 - The decision of whether to estimate the value function or its derivative is often overlooked in the dynamic programming literature, especially within the operations research community. In the controls community, use of gradients is sometimes referred to as *dual heuristic dynamic programming* (see Werbos (1992) and Venayagamoorthy et al. (n.d.)).

Section 19.1 - The theory behind the projective SPAR algorithm is given in Powell et al. (2004). A proof of convergence of the leveling algorithm is given in Topaloglu & Powell (2003).

Section 19.9 - The first paper to formulate a math program with uncertainty appears to be Dantzig & Ferguson (1956). For a broad introduction to the field of stochastic optimization, see Ermoliev (1988) and Pflug (1996). For complete treatments of the field of stochastic programming, see **?**, Shapiro (2003), Birge & Louveaux (1997), and Kall & Mayer (2005). For an easy tutorial on the subject, see Sen & Higle (1999). A very thorough introduction to stochastic programming is given in Ruszczyński & Shapiro (2003). Mayer (1998) provides a detailed presentation of computational work for stochastic programming. There has been special interest in the types of network problems we have considered (see **?**, Wallace (1986) and **?**). Rockafellar & Wets (1991) presents specialized algorithms for stochastic programs formulated using scenarios. This modeling framework has been of particular interest in the are of financial portfolios (Mulvey & Ruszczyński (1995)). Benders' decomposition for two-stage stochastic programs was first proposed by **?** as the "L-shaped" method. Higle & Sen (1991) introduce stochastic decomposition, which is a Monte-Carlo based algorithm that is most similar in spirit to approximate dynamic programming. Chen & Powell (1999) present a variation of Benders that falls between stochastic decomposition and the L-shaped method. The relationship between Benders' decomposition and dynamic programming is often overlooked. A notable exception is **?**, which uses Benders to solve a resource allocation problem arising in the management of reservoirs. This paper presents Benders as a method for avoiding the curse of dimensionality of dynamic programming. For an excellent review of Benders' decomposition for multistage problems, see Ruszczyński (2003). Benders has been extended to multistage problems in Birge (1985), Ruszczyński (1993), and Chen & Powell (1999), which can be viewed as a form of approximate dynamic programming using cuts for value function approximations.

Section 19.10.1 - The proof of the projection operation is based on Powell et al. (2004).

PROBLEMS

19.1 Consider a newsvendor problem where we solve

$$\max_x \mathbb{E}F(x, \hat{D}),$$

where

$$F(x, \hat{D}) = p \min(x, \hat{D}) - cx.$$

We have to choose a quantity x before observing a random demand $\hat{D}$. For our problem, assume that $c = 1$, $p = 2$, and that $\hat{D}$ follows a discrete uniform distribution between 1 and 10 (that is, $\hat{D} = d, d = 1, 2, \ldots, 10$ with probability 0.10). Approximate $\mathbb{E}F(x, \hat{D})$ as a piecewise linear function using the methods described in section 19.1, using a stepsize $\alpha_{n-1} = 1/n$. Note that you are using derivatives of $F(x, \hat{D})$ to estimate the slopes of the function. At each iteration, randomly choose x between 1 and 10. Use sample realizations of the gradient to estimate your function. Compute the exact function and compare your approximation to the exact function.

19.2 Repeat exercise 19.1, but this time approximate $\mathbb{E}F(x, \hat{D})$ using a linear approximation:

$$\overline{F}(x) = \theta x.$$

Compare the solution you obtain with a linear approximation to what you obtained using a piecewise-linear approximation. Now repeat the exercise using demands that are uniformly distributed between 500 and 1000. Compare the behavior of a linear approximation for the two different problems.

19.3 Repeat exercise 19.1, but this time approximate $\mathbb{E}F(x, \hat{D})$ using the SHAPE algorithm. Start with an initial approximation given by

$$\overline{F}^0(x) = \theta_0(x - \theta_1)^2.$$

Use the recursive regression methods of sections 19.7 and 3.8 to fit the parameters. Justify your choice of stepsize rule. Compute the exact function and compare your approximation to the exact function.

19.4 Repeat exercise 19.1, but this time approximate $\mathbb{E}F(x, \hat{D})$ using the regression function given by

$$\overline{F}(x) = \theta_0 + \theta_1 x + \theta_2 x^2.$$

Use the recursive regression methods of sections 19.7 and 3.8 to fit the parameters. Justify your choice of stepsize rule. Compute the exact function and compare your approximation to the exact function. Estimate your value function approximation using two methods:

(a) Use observations of $F(x, \hat{D})$ to update your regression function.

(b) Use observations of the derivative of $F(x, \hat{D})$, so that $\overline{F}(x)$ becomes an approximation of the derivative of $\mathbb{E}F(x, \hat{D})$.

19.5 Approximate the function $\mathbb{E}F(x, \hat{D})$ in exercise 19.1, but now assume that the random variable $\hat{D} = 1$ (that is, it is deterministic). Using the following approximation strategies:

(a) Use a piecewise linear value function approximation. Try using both left and right derivatives to update your function.

(b) Use the regression $\overline{F}(x) = \theta_0 + \theta_1 x + \theta_2 x^2$.

19.6 We are going to solve the basic asset acquisition problem (section 8.2.2) where we purchase assets (at a price p^p) at time t to be used in time interval $t + 1$. We sell assets at a price p^s to satisfy the demand $\hat{D}_t$ that arises during time interval t. The problem is to be solved over a finite time horizon T. Assume that the initial inventory is 0 and that demands follow a discrete uniform distribution over the range $[0, D^{max}]$. The problem parameters are given by

$$\begin{aligned} \gamma &= 0.8, \\ D^{max} &= 10, \\ T &= 20, \\ p^p &= 5, \\ p^s &= 8. \end{aligned}$$

Solve this problem by estimating a piecewise linear value function approximation (section 19.1). Choose $\alpha_{n+1} = a/(a+n)$ as your stepsize rule, and experiment with different values of a (such as 1, 5, 10, and 20). Use a single-pass algorithm, and report your profits (summed over all time periods) after each iteration. Compare your performance for different stepsize rules. Run 1000 iterations and try to determine how many iterations are needed to produce a good solution (the answer may be substantially less than 1000).

19.7 Repeat exercise 19.6, but this time use the SHAPE algorithm to approximate the value function. Use as your initial value function approximation the function

$$\overline{V}_t^0(R_t) \quad = \quad \theta_0(R_t - \theta_2)^2.$$

For each of the exercises below, you may have to tweak your stepsize rule. Try to find a rule that works well for you (we suggest stick with a basic $a/(a+n)$ strategy). Determine an appropriate number of training iterations, and then evaluate your performance by averaging results over 100 iterations (testing iterations) where the value function is not changed.

(a) Solve the problem using $\theta_0 = 1, \theta_1 = 5$.

(b) Solve the problem using $\theta_0 = 1, \theta_1 = 50$.

(c) Solve the problem using $\theta_0 = 0.1, \theta_1 = 5$.

(d) Solve the problem using $\theta_0 = 10, \theta_1 = 5$.

(e) Summarize the behavior of the algorithm with these different parameters.

19.8 Repeat exercise 19.6, but this time assume that your value function approximation is given by

$$\overline{V}_t^0(R_t) = \theta_0 + \theta_1 R_t + \theta_2 R_t^2.$$

Use the recursive regression techniques of sections 19.7 and 3.8 to determine the values for the parameter vector θ.

19.9 Repeat exercise 19.6, but this time assume you are solving an infinite horizon problem (which means you only have one value function approximation).

19.10 Repeat exercise 19.8, but this time assume an infinite horizon.

19.11 Repeat exercise 19.6, but now assume the following problem parameters:

$$\begin{aligned} \gamma &= 0.99, \\ T &= 200, \\ p^p &= 5, \\ p^s &= 20. \end{aligned}$$

For the demand distribution, assume that $\hat{D}_t = 0$ with probability 0.95, and that $\hat{D}_t = 1$ with probability 0.05. This is an example of a problem with low demands, where we have to hold inventory for a fairly long time.

CHAPTER 20

LOOKAHEAD POLICIES

Up to now we have considered three classes of policies: policy function approximations and parametric cost function approximations, both of which need to be tuned using policy search, and policies that depend on value function approximations which approximate the impact of a decision on the future through the state variable. All three of these policies depend on approximating some function, which means we are limited by our ability to approximate the function.

Not surprisingly, we cannot always develop sufficiently accurate functional approximations. Policy function approximations have been most successful when decisions are low-dimensional, continuous controls (for example, you would never use a PFA to dispatch locomotives for a railroad). Similarly, approximating the value of being in a state works very well when problems are simple, or when we can exploit problem structure (as we did in chapter 19 when we used the convexity of the problem), but there are many problems where the value function is simply too complex.

When all else fails (and it often does), we have to resort to direct lookahead policies (DLAs), which optimize over some horizon to help capture the impact of decisions now on the future, from which we can extract the decision we would make now. This is a much more brute force approach and, not surprisingly, is typically very hard computationally. As a result, the challenge here is introducing approximations that make this problem tractable.

20.1 OPTIMAL POLICIES USING LOOKAHEAD MODELS

Lookahead policies are best described by restating our objective function

$$F(S_0) = V_0(S_0) = \max_{\pi\in\Pi} \mathbb{E}^\pi \left\{ \sum_{t'=0}^{T} C(S_{t'}, X^\pi_{t'}(S_{t'}))|S_0 \right\} \tag{20.1}$$

Now imagine solving this starting at time t:

$$V_t(S_t) = \max_{\pi\in\Pi} \mathbb{E}^\pi \left\{ \sum_{t'=t}^{T} C(S_{t'}, X^\pi_{t'}(S_{t'}))|S_t \right\} \tag{20.2}$$

Equation (20.2) is Bellman's optimality equation, which is nothing more than the definition of the value of starting in a state S_t and following an optimality policy.

Since the decision x_t is a deterministic function of the state S_t, we can rewrite (20.2) as

$$V_t(S_t) = \max_{x_t} \left(C(S_t, x_t) + \mathbb{E} \left\{ \max_{\pi\in\Pi} \mathbb{E}^\pi \left\{ \sum_{t'=t+1}^{T} C(S_{t'}, X^\pi_{t'}(S_{t'}))|S_{t+1} \right\} |S_t, x_t \right\} \right) \tag{20.3}$$

Finally, we can write this as a policy

$$\begin{aligned} X^{LA}_t(S_t) &= \arg\max_{x_t} \left(C(S_t, x_t) + \mathbb{E} \left\{ \min_{\pi\in\Pi} \mathbb{E}^\pi \left\{ \sum_{t'=t+1}^{T} C(S_{t'}, X^\pi_{t'}(S_{t'}))|S_{t+1} \right\} |S_t, x_t \right\} \right) &(20.4)\\ &= \arg\max_{x_t} \left(C(S_t, x_t) + \mathbb{E}\{V_{t+1}(S_{t+1})|S_t, x_t\} \right) &(20.5)\\ &= \arg\max_{x_t} \left(C(S_t, x_t) + V^x_t(S^x_t) \right). &(20.6) \end{aligned}$$

Equation (20.6) is the basic statement of a lookahead policy, where we make a decision now while optimizing over the entire horizon. Needless to say, computing this policy is computationally intractable for all but a small class of problems (such as decision trees, which we first saw in section 2.1.4, but which we revisit below). If we could solve equation (20.6), that is like saying that we can solve the original problem (20.1) directly. If that were possible, we are done.

In chapters 17-18 (and chapter 19 for convex problems), we pursued a strategy of replacing $V_{t+1}(S_{t+1})$ (or the post-decision version $V^x_t(S^x_t)$) with an approximation $\overline{V}_{t+1}(S_{t+1})$ (or $\overline{V}^x_t(S^x_t)$). The difficulty is that there are many applications where it is simply not possible to obtain high quality approximations of the value of being in a state.

Some examples of problems where the value of the future is not easy to approximate include

- Problems with complex interactions - Imagine a stochastic scheduling problem (routing vehicles, scheduling machines, scheduling doctors) which involve complex interactions in the future. To make a decision now (for example, to commit to serve a job or patient in the future), it is necessary to explicitly plan the schedule in the future.

- Problems with forecasts - Consider a problem of managing inventories of products over a holiday, where we have a forecast $f_t = (f_{tt'})_{t'\geq t}$ for the demands. Since forecasts evolve over time, they should be a part of the state variable, but this is never done. Forecasts can be modeled as latent (hidden) variables (as we discuss below), but they are more naturally handled in a lookahead model.

- Multilayer resource allocation problems - Value functions can be effective when modeling single layer resource allocation problems (managing water, blood, money), but many problems involve multiple layers (jobs and machines, trucks and packages, blood and patients). It is very hard to capture the value of being in a state that includes more than one resource layer.

Lookahead models are widely used, and as such have evolved in different communities under different names, including

Rolling horizon procedure Often used in operations research, it refers to the process of optimizing over an interval $(t, t + H)$, implementing decisions for time t, rolling to $t + 1$ (and sampling/observing new information), and then solving over the interval $(t + 1, t + H + 1)$ (hence the name "rolling horizon").

Receding horizon procedure This is a term often used in computer science, but means the same as rolling horizon procedure.

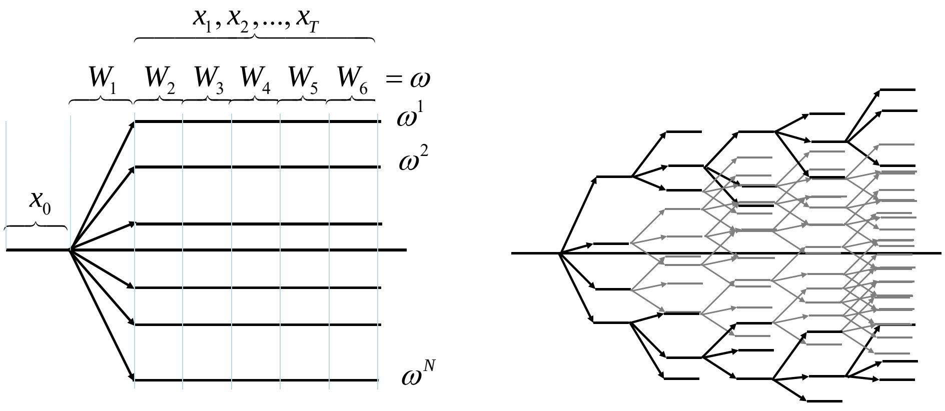
Model predictive control This is the term used in the engineering-controls community, and refers to the fact that if we create a lookahead model, we need an explicit model of the problem. It is quite common in engineering that we do not have such a model, which limits the use of lookahead models (we might be able to create an approximation). The controls literature in engineering focuses mostly on deterministic problems, and as a result, MPC (the standard abbreviation for model predictive control) is typically associated with deterministic models of the future. However, the term "model predictive control" technically applies to any model-based approximation of the future, and they may be deterministic (which is most common) or stochastic.

20.2 STRATEGIES FOR APPROXIMATING THE LOOKAHEAD MODEL

There are a variety of strategies that we can use for approximating the lookahead model to make solving (20.6) computationally intractable. These include

Limiting the horizon - We may reduce the horizon from (t, T) to $(t, t + H)$, where H is a suitable short horizon that is chosen to capture important behaviors. For example, we might want to model water reservoir management over a 10 year period, but a lookahead policy that extends one year might be enough to produce high quality decisions. We can then simulate our policy to produce forecasts of flows over all 10 years.

Stage aggregation - A stage represents the process of revealing information followed by the need to make a decision. A common approximation is a two-stage formulation (see Figure 20.1(a)), where we make a decision x_t, then observe all future events (until $t + H$), and then make all remaining decisions. By contrast, a multistage formulation would explicitly model the sequence: decision, information, decision, information, and so on. Figure 20.1(b) illustrates the many possible paths in a multistage formulation, which generally make these formulations computationally intractable.



(a) (b)

Figure 20.1 Illustration of (a) a two-stage scenario tree and (b) a multistage scenario tree.

Outcome aggregation or sampling - Instead of using the full set of outcomes Ω (which is often infinite), we can use Monte Carlo sampling to choose a small set of possible outcomes that start at time t (assuming we are in state S_t^n during the nth simulation through the horizon) through the end of our horizon $t + H$. We refer to this as $\tilde{\Omega}_t^n$ to capture that it is constructed for the decision problem at time t while in state S_t^n. The simplest model in this class is a deterministic lookahead, which uses a single point estimate.

Discretization - Time, states, and decisions may all be discretized in a way that makes the resulting model computationally tractable. In some cases, this may result in a Markov decision process that may be solved exactly using backward dynamic programming (which we introduced in chapter 14). Because the discretization generally depends on the current state S_t, this model will have to be solved all over again after we make the transition from t to $t + 1$.

Dimensionality reduction - We may ignore some variables in our lookahead model as a form of simplification. For example, a forecast of weather or future prices can add a number of dimensions to the state variable. While we have to track these in the base model (including the evolution of these forecasts), we can hold them fixed in the lookahead model, and then ignore them in the state variable (these become *latent variables*).

Of these, the most subtle is the last one, where we introduce simplifications by holding some variables constant in the lookahead model that actually change (randomly) in the base model (or the real world, if our policy is being tested in an online setting). One example is a forecast, where it is common to hold a forecast constant over the horizon (although it may vary over time within the horizon), while in fact forecasts evolve randomly over time.

Given these approximations, it is important to make a distinction between the *lookahead model*, and the *base model* which we are trying to optimize with our lookahead policy. We begin by noting that a lookahead model has to be indexed by the time t at which it is being formulated. Since it extends over a horizon from t to $\min\{t + H, T\}$, we index every variable by t (which fixes the information content of the model) and t', which is the time period within the lookahead horizon.

Then, we suggest using the same variables as in the base model, but with a "tilde." Thus, $\tilde{S}_{tt'}$ would be the state in our lookahead model at time t' within the lookahead horizon, for a model being solved at time t. $\tilde{S}_{tt'}$ might have fewer variables than S_t (or $S_{t'}$), and we might also use a different level of aggregation. Using this notation, our lookahead policy would be written

$$X_t^{LA}(S_t) \quad = \quad \arg\max_{x_t} \left(C(S_t, x_t) + \mathbb{E} \left\{ \max_{\tilde{\pi} \in \tilde{\Pi}} \tilde{\mathbb{E}}^{\pi} \left\{ \sum_{t'=t+1}^{t+H} C(\tilde{S}_{tt'}, \tilde{X}_{tt'}^{\pi}(\tilde{S}_{tt'})) | \tilde{S}_{t,t+1} \right\} | S_t, x_t \right\} \right) \tag{20.7}$$

where $\tilde{S}_{t,t'+1} = \tilde{S}^M(\tilde{S}_{tt'}, \tilde{X}_t^{\tilde{\pi}}(\tilde{S}_{tt'}), \widetilde{W}_{t,t'+1})$ describes the dynamics within our lookahead model, and where $\tilde{X}_t^{\tilde{\pi}}(\tilde{S}_{tt'})$ is the policy corresponding to $\tilde{\pi}$.

Here, we write $\tilde{\Pi}$ as a modified set of policies, and $\tilde{\mathbb{E}}$ as a modified set of random outcomes. We might even be modeling time differently (e.g. hourly time steps instead of 5 minutes), but we are going to keep the same time notation for simplicity.

The remainder of this chapter describes different strategies that have been used for approximating lookahead models.

20.3 LOOKAHEAD MODELS WITH DISCRETE ACTIONS

We begin with problems with discrete actions, where we assume that there are no more than perhaps 100 actions possible from each state (and 100 is reasonably large). Our methods are actually quite robust with respect to complex state variables and forms of uncertainty, but we need to keep the action spaces reasonable.

Problems with discrete action spaces represents a very important class of problems by themselves. At the same time, we will be illustrating methods that we can draw on when decisions are vectors.

20.3.1 A deterministic lookahead: shortest path problems

The best way to illustrate a lookahead policy is the process of finding the best path over a transportation network where travel times are evolving randomly as traffic moves. Imagine that we are trying to get from an origin q to a destination r, and assume that we are at an intermediate node i (trying to get to r). Our navigation system will recommend that we go from i to some node j by first finding the shortest path from i to r, and then using this path to determine what to do now.

This problem is solved as a deterministic (but time-dependent) dynamic programming problem. To simplify the notation, we are going to assume that each movement over a link (i, j) takes one time period. We represent our traveler only when he is at a node, since this is the only time when there is a real decision. Imagine that it is time t, and that we are at node q heading to r. Define

c_{tij} = The estimated cost, made at time t, of traversing link (i, j),

x_{tij} = The flow that we plan, at time t, on traversing link (i, j) (typically at some time in the future.)

In our shortest path problem, the flow x_{tij} is either 1, meaning that link (i, j) is in the shortest path from q to r, and 0 otherwise.

Assume that we are sure we can arrive by time T, but if we arrive earlier, then we do nothing at node r until time T. We can write our problem as

$$\min_{x_t,\dots,X_T} \sum_{t'=t}^{T} \sum_i \sum_j c_{tij} x_{tij} \tag{20.8}$$

subject to flow conservation constraints:

$$\sum_j x_{tqj} \quad = \quad 1, \tag{20.9}$$

$$\sum_k x_{t',ki} - \sum_j x_{t'+1,ij} \quad = \quad 0, t' = t,\dots,T-1,\ \forall i, \tag{20.10}$$

$$\sum_i x_{T-1,ir} \quad = \quad 1. \tag{20.11}$$

The optimization model (20.8)-(20.11) is a lookahead model that is optimizing the problem from time t until the end of the horizon T. Constraint (20.9) specifies that one unit of flow has to go out of origin node q at time t. Constraints (20.10) ensure that flow into each intermediate node is equal to the flow out. Finally, constraint (20.11) ensures that there is one unit of flow into node r at time T.

Shortest path problems are always solved as (highly specialized) dynamic programs. When combined with some careful software engineering, problem (20.8)-(20.11) can be easily solved using Bellman's equation (for deterministic problems):

$$V_{t'i} = \min_j \big(c_{t'ij} + V_{t'+1,j}\big), \tag{20.12}$$

for $t' = t,\dots,T$ and all nodes i.

Both the linear program (20.8)-(20.11) and the deterministic dynamic program (20.12) represent deterministic lookahead models. When we solve the linear program, all we use is the decision x_t that tells us what to do at time t. Similarly, we use the decision from our dynamic program

$$x_t^* = \arg\min_j \big(c_{tqj} + V_{t+1,j}\big)$$

which tells us which node j we should go to. If $x_{tqj} = 1$, we will reoptimize when we arrive at node j at time $t+1$, at which time the costs may have changed.

It does not matter whether we are solving the linear program (20.8)-(20.11) or the deterministic dynamic program (20.12), both methods are solving a deterministic approximation of the future, because our real problem is, of course, stochastic. For example, if we encounter unexpected congestion, we will re-optimize (but again using a deterministic approximation).

20.3.2 Lookahead dynamic programs

It is important not to overlook the potential of solving the lookahead model as a dynamic program using the modeling and algorithmic framework of Markov decision processes which we introduced in chapter 14. Here, we take advantage of the opportunity to introduce a variety of approximations in the lookahead model that might not exist in the base model. The state S_t in the base model might be multidimensional and continuous. For example, there are many problems that feature rolling forecasts $f_t = (f_{tt'})_{t' \geq t}$, which is clearly multidimensional. Yet, we might solve a lookahead model at time t using a fixed set of forecasts which we hold constant over the horizon of the lookahead model. In this case, the forecast would be a *latent variable* in the lookahead dynamic program.

In addition to ignoring variables, there are other ways of approximating a lookahead model that might open the door to using the methods of Markov decision processes:

- Discretizing (or more coarsely discretizing) different dimensions of the state variable.
- Using a coarser discretization of time.
- Simplifying the exogenous information process (discretization, ignoring one or more dimensions, using a sampled representation).
- Simplifying the decision variable (discretization, clustering vectors into a smaller, discrete set of actions).

It is not uncommon for a paper to formulate and solve a Markov decision problem where these approximations have already been made. The resulting MDP may still be hard to solve - even reducing a problem to a discretized five-dimensional state variable will still require the use of approximation methods. As a result, it is possible to become so focused on solving the approximate MDP that we lose sight of the fact that the resulting model is still just an approximate lookahead model. We return to this issue at the end of the chapter.

20.3.3 Decision trees

One of the most effective ways of communicating the process of making decisions under uncertainty is to use decision trees. Figure 20.2 illustrates a problem facing a Little League baseball coach trying to schedule a playoff game under the threat of bad weather. The coach first has to decide if he should check the weather report. Then he has to decide if he should schedule the game. Bad weather brings a poor turnout that reduces revenues from tickets and the concession stand. There are costs if the game is scheduled (umpires, food, people to handle parking and the concession stand) that need to be covered by the revenue the game might generate.

In figure 20.2, squares denote decision nodes where we have to choose an action (Does he check the weather report? Does he schedule the game?), while circles represent outcome nodes where new (and random) information arrives (What will the weather report say? What will the weather be?). We can "solve" the decision tree (that is, find the best decision given the information available), by rolling backward through the tree. In figure 20.3(a), we have found the expected value of being at each of the end outcome nodes. For example, if we check the weather report and see a forecast of rain, the probability it will actually rain is 0.80, producing a loss of \$2000; the probability that it will be cloudy is 0.20, producing a profit of \$1000; the probability it will be sunny is zero (if it were sunny, we would make a profit of \$5000). The expected value of scheduling the game, when the weather forecast is rain, is $(0.80)(-\$2000) + (0.20)(\$1000) + (0)(\$5000) = -\$1400$. Repeating this calculation for each of the ending outcome nodes produces the results given in figure 20.3(a).

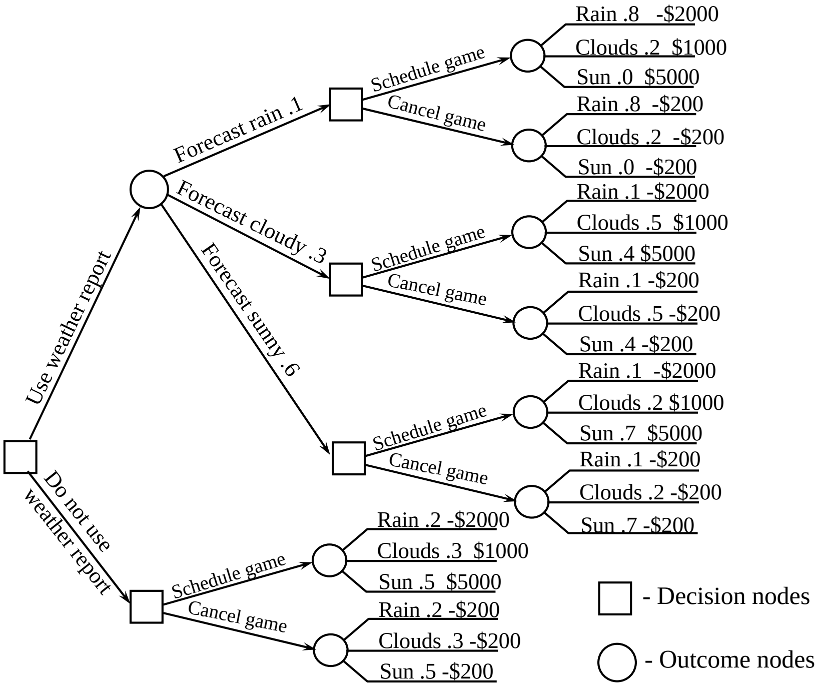


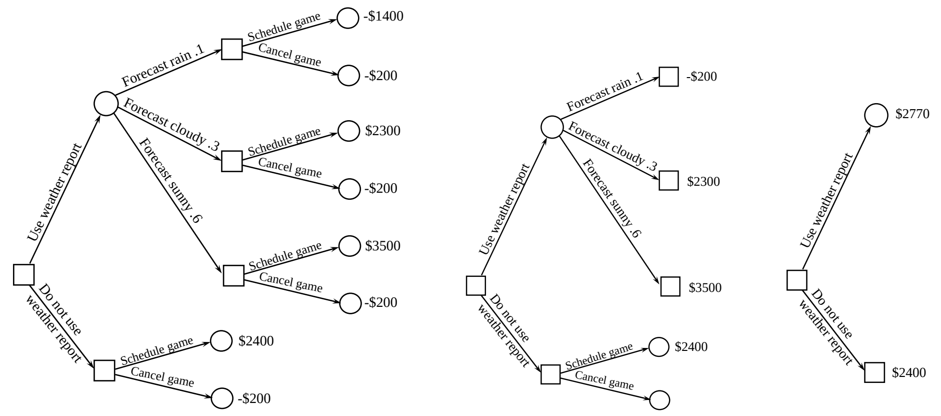
Figure 20.2 Decision tree showing decision nodes and outcome nodes.

At a decision node, we get to choose an action, and of course we choose the action with the highest expected profit. The results of this calculation are given in figure 20.3(b). Finally, we have to determine the expected value of checking the weather report by again by multiplying the probability of each possible weather forecast (rainy, cloudy, sunny) times the expected value of each outcome. Thus, the expected value of checking the weather report is $(.1)(-\$200)+(.3)(\$2300)+(.6)(\$3500) = \$2770$, shown in figure 20.3(c). The expected value of making decisions without the weather report is $2400, so the analysis shows that we should check the weather report. Alternatively, we can interpret the result as telling us that we would be willing to pay up to $300 for the weather report.

Almost any decision problem with discrete states and actions can be modeled as a decision tree. The problem is that they are not practical when there is a large number of actions, as well as a large number of information outcomes. Even when these are not too large, decision trees still grow exponentially, sharply limiting the number of time periods that can be modeled. We return to figure 20.4, that we first saw in chapter 14 that illustrates how quickly decision trees explode, even for relatively small problems.

20.3.4 Monte Carlo tree search

For problems where the number of actions per state is not too large (but where the set of random outcomes may be quite large), we may replace the explicit enumeration of the entire tree with a heuristic policy to evaluate what might happen after we reach a state.



20.3(a) 20.3(b) 20.3(c)

Figure 20.3 Evaluating a decision tree. (a) Evaluating the final outcome nodes. (b) Evaluating the final decision nodes. (c) Evaluating the first outcome nodes.

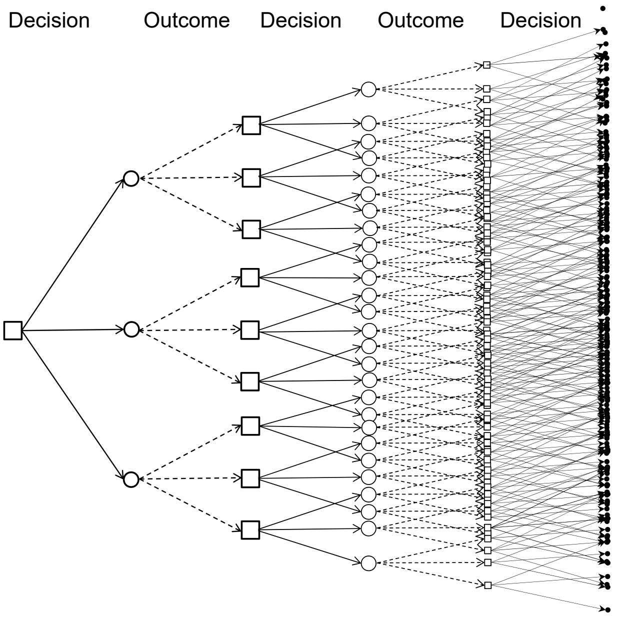


Figure 20.4 Decision tree illustrating the sequence of decisions and new information, illustrating the explosive growth of decision trees.

Imagine that we are trying to evaluate if we should take action a_0 which takes us to state S_0^a (the post-decision state), after which we choose at random an outcome of W_1, which puts us in the next (pre-decision) state S_1. If we repeat this process for each action a_0, and then for each action a_1 out of each of the downstream states S_1, the tree would explode in size.

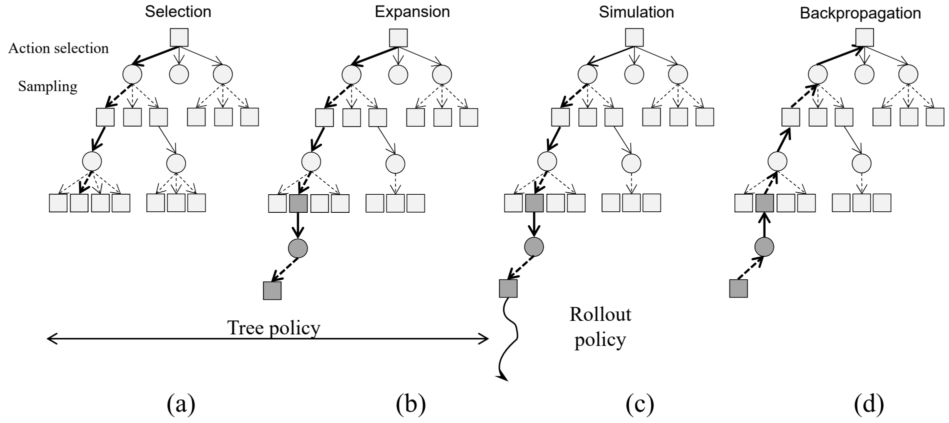


Figure 20.5 Illustration of Monte Carlo tree search, illustrating (left to right): selection, expansion, simulation and backpropagation.

Monte Carlo tree search is a technique that originated in computer science, where it has been primarily used for deterministic problems. MCTS (as it is widely known) proceeds by applying a simple test to each action out of a node, and then using this test to choose one action to explore. This may result in a traversal to a state we have visited before, at which point we simply repeat the process, or we may find ourselves at a new state. We then call a *roll out policy* which is some simple rule for making decisions that depends on the problem at hand. The rollout policy gives us a rough estimate of the value of being in this new state. If the state is attractive enough, it is added to the tree.

Each node (state) in the tree is described by four quantities:

1) The pre-decision value function $\widetilde{V}_{tt'}(\tilde{S}_{tt'})$, the post-decision value function $\widetilde{V}^a_{tt'}(\tilde{S}^a_{tt'})$, and the contribution $C(\tilde{S}_{tt'}, \tilde{a}_{tt'})$ from being in state $\tilde{S}_{tt'}$ and taking action $\tilde{a}_{tt'}$.

2) The visit count, $N(\tilde{S}_{tt'})$, which counts the number of times we have performed rollouts (explained below) from state $\tilde{S}_{tt'}$.

3) The count, $N(\tilde{S}_{tt'}, \tilde{a}_{tt'})$, which counts the number of times we have taken decision $\tilde{a}_{tt'}$ from state $\tilde{S}_{tt'}$.

4) The set of actions $\mathcal{A}_s$ from each state s and the random outcomes $\tilde{\Omega}_{t,t'+1}(\tilde{S}^a_{tt'})$ that might happen when in post-decision state $\tilde{S}^a_{tt'}$.

Monte Carlo tree search progresses in four steps which are illustrated in figure 20.5, where the detailed steps of the MCTS are described in a series of procedures. Note that, as before, we let $\tilde{S}_{tt'}$ be the pre-decision state, which is the node that precedes a decision. A deterministic function $S^{M,a}(\tilde{S}_{tt'}, \tilde{a}_{tt'})$ takes us to a post-decision state $\tilde{S}^a_{tt'}$, after which a Monte Carlo sample of the exogenous information takes us to the next pre-decision state $\tilde{S}_{t,t'+1}$.

1) Selection There are two steps in the selection phase. The first (and most difficult) requires choosing an action, while the second involves taking a Monte Carlo sample of any random information.

```
function $MCTS(S_t)$
Step 0. Create root note $\tilde{S}_{tt} = S_t$; set iteration counter $n = 0$.
Step 1. while $n < n^{thr}$
        Step 1.1 $\tilde{S}_{tt'} \leftarrow TreePolicy(\tilde{S}_{tt})$
        Step 1.2 $\widetilde{V}_{tt'}(\tilde{S}_{tt'}) \leftarrow SimPolicy(\tilde{S}_{tt'})$
        Step 1.3 $Backup(\tilde{S}_{tt'}, \widetilde{V}_{tt'}(\tilde{S}_{tt'}))$
        Step 1.4 $n \leftarrow n + 1$
Step 2. $\tilde{a}^*_t = \arg\max_{\tilde{a}_{tt} \in \tilde{\mathcal{A}}_{tt}(\tilde{S}_{tt})} \tilde{C}(\tilde{S}_{tt}, \tilde{a}_{tt}) + \widetilde{V}^a_{tt}(\tilde{S}^a_{tt})$
Step 3. return $a^*_t$.
```

Figure 20.6 Sampled MCTS algorithm.

1a) Choosing the action The first step from any node (already generated) is to choose an action (see figure 20.5(a) and the algorithm in figure 20.6). The most popular policy for choosing an action is to use a type of upper confidence bound (recall we introduced UCB policies in section 7.3.2) adapted for trees, hence its name, Upper Confidence bounding for Trees (UCT). We could simply choose the action that appears to be best, but we could get stuck in a solution were we avoid actions that do not look attractive. Since our estimates are only approximations, we have to recognize that we may not have explored them enough (the classic exploration-exploitation tradeoff). In this setting, the UCT policy is given by

$$A^{UCT}_{tt'}(\tilde{S}_{tt'}|\theta^{UCT}) = \arg\max_{\tilde{a} \in \tilde{\mathcal{A}}_{tt'}} \left(\left(C(\tilde{S}_{tt'}, \tilde{a}) + \widetilde{V}^a_{tt'}(\tilde{S}^a_{tt'})\right) + \theta^{UCT} \sqrt{\frac{\ln N(\tilde{S}_{tt'})}{N(\tilde{S}_{tt'}, \tilde{a}_{tt'})}} \right)$$

The parameter θ^{UCT} has to be tuned, just as we would tune any policy. As with UCB policies, the square root term is designed to encourage exploration, by putting a bonus for actions that have not been explored as often. A nice feature of UCT policies is that they are very easy to compute, which is important in an MCTS setting where we need to quickly evaluate many actions.

1b) Sampling the outcome Here we assume that we can simply take a Monte Carlo sample of any random information (see section 10.4). There are settings where simple Monte Carlo sampling is not very efficient, such as when the random outcome might be a success or failure, where one or the other dominates.

2) Expansion If the action we choose above is one we have chosen before, then we progress to the next post-decision state (the solid line connecting the square node to the round node in figure 20.5(b)) at which point when then sample another random outcome which brings us to a new pre-decision state (see the algorithm in figure 20.7). But if we have not chosen this action before, then we expand our tree by first adding the link associated with the decision to the post-decision state node, followed by a Monte Carlo sample which takes us to the subsequent pre-decision state. At this point we have to deal with the fact that we would not have an estimate of the value of being in this state (which we need for our UCT policy). To overcome this, we call our simulation policy, which is a form of roll-out policy (discussed next).

```
function $TreePolicy(\tilde S_{tt})$
Step 0. $t' \leftarrow t$
Step 1. while $\tilde S_{tt'}$ is non-terminal do
        Step 2. if $|\tilde{\mathcal{A}}_{tt'}(\tilde S_{tt'})| < d^{thr}$ do (Expanding a decision out of a pre-decision state)
                Step 2.1 Choose decision $\tilde a^*_{tt'}$ by optimizing on the basis of the contribution of the decision
                    $\tilde C(\tilde S_{tt'}, \tilde a_{tt'})$, then taking a Monte Carlo sample to the next pre-decision state $\tilde S_{t,t'+1}$, and
                    then finally using the rollout policy to approximate the value of being in state $\tilde S_{t,t'+1}$.
                Step 2.2 $\tilde S^a_{tt'} = S^M(\tilde S_{tt'}, \tilde a^*_{tt'})$ (Expansion step)
                Step 2.3 $\tilde{\mathcal{A}}_{tt'}(\tilde S_{tt'}) \leftarrow \tilde{\mathcal{A}}_{tt'}(\tilde S_{tt'}) \bigcup \{\tilde a^*_{tt'}\}$
                Step 2.4 $\tilde{\mathcal{A}}^u_{tt'}(\tilde S_{tt'}) \leftarrow \tilde{\mathcal{A}}^u_{tt'}(\tilde S_{tt'}) - \{\tilde a^*_{tt'}\}$
        else  Step 2.5 $\tilde a^*_{tt'} = \arg\max_{\tilde a_{tt'} \in \tilde{\mathcal{A}}_{tt'}(\tilde S_{tt'})} \left( \left( \tilde C(\tilde S_{tt'}, \tilde a_{tt'}) + \widetilde V^a_{tt'}(\tilde S^a_{tt'}) \right) + \theta^{UCT} \sqrt{\frac{\ln N(\tilde S_{tt'})}{N(\tilde S_{tt'}, \tilde a_{tt'})}} \right)$
                Step 2.6 $\tilde S^a_{tt'} = S^M(\tilde S_{tt'}, \tilde a^*_{tt'})$
        end if
        Step 3 if $|\tilde\Omega_{t,t'+1}(\tilde S^a_{tt'})| < e^{thr}$ do (Expanding an exogenous outcome out of a post-decision state)
                Step 3.1 Choose exogenous event $\widetilde W_{t,t'+1}$,
                Step 3.2 $\tilde S_{t,t'+1} = S^{M,a}(\tilde S^a_{tt'}, \widetilde W_{t,t'+1})$ (Expansion step)
                Step 3.3 $\tilde\Omega_{t,t'+1}(\tilde S^a_{tt'}) \leftarrow \tilde\Omega_{t,t'+1}(\tilde S^a_{tt'}) \bigcup \{\widetilde W_{t,t'+1}\}$
                Step 3.4 $\tilde\Omega^u_{t,t'+1}(\tilde S^a_{tt'}) \leftarrow \tilde\Omega^u_{t,t'+1}(\tilde S^a_{tt'}) - \{\widetilde W_{t,t'+1}\}$
                Step 3.5 $t' \leftarrow t' + 1$
                return $\tilde S_{tt'}$ (stops execution of while loop)
        else  Step 3.6 Choose exogenous event $\widetilde W_{t,t'+1}$,
                Step 3.7 $\tilde S_{t,t'+1} = S^{M,a}(\tilde S^a_{tt'}, \widetilde W_{t,t'+1})$
                Step 3.8 $t' \leftarrow t' + 1$
                end if
        end while
```

Figure 20.7 The tree policy.

3) Simulation The simulation step assumes we have access to some policy which is easy to execute that allows us to obtain a quick and reasonable estimate of the value of being in a state (see figure 20.5(c) and the algorithm in figure 20.8). Of course, this is very problem dependent. Some strategies include:

- A myopic policy, which greedily makes choices. There are problems where myopic policies are reasonable starting estimates (of course they are suboptimal). However, such a greedy policy can be extremely poor (imagine finding the shortest path through a network by always choosing the shortest link out of a node).
- A parameterized policy with reasonable estimates of the parametres. We might have a rule for selling an asset if its price rises by some percentage. Such a rule will not be ideal, but it will be reasonable.
- A posterior bound. We might sample all future information, and then make the best decision assuming that this future information comes true.

```
**function** $SimPolicy(\tilde{S}_{tt'})$
**Step 0.** Choose a sample path $\tilde{\omega} \in \tilde{\Omega}_{tt'}$
**Step 1.** **while** $\tilde{S}_{tt'}$ is non-terminal
        **Step 2.1** Choose $\tilde{a}_{tt'} \leftarrow \tilde{\pi}(\tilde{S}_{tt'})$ where $\tilde{\pi}$ is the rollout policy.
        **Step 2.2** $\tilde{S}_{t,t'+1} \leftarrow S^M(\tilde{S}_{tt'}, \tilde{a}_{tt'}(\tilde{\omega}))$
        **Step 2.3** $t' \leftarrow t' + 1$
**end while**
**return** $\widetilde{V}_{tt'}(\tilde{S}_{tt'})$ (Value function of $\tilde{S}_{tt'}$)
```

Figure 20.8 This function simulates the policy.

```
**function** $Backup(\tilde{S}_{tt'}, \widetilde{V}_{tt'}(\tilde{S}_{tt'}))$
**while** $\tilde{S}_{tt'}$ is not null **do**
        **Step 1.1** $N(\tilde{S}_{tt'}) \leftarrow N(\tilde{S}_{tt'}) + 1$
        **Step 1.2** $t^* \leftarrow$ t'-1.
        **Step 1.3** $N(\tilde{S}_{t,t^*-1}, \tilde{a}_{t,t^*-1}) \leftarrow N(\tilde{S}_{t,t^*-1}, \tilde{a}_{t,t^*-1}) + 1$
        **Step 1.4** $\widetilde{V}^a_{t,t^*-1}(\tilde{S}^a_{t,t^*-1}) \leftarrow \frac{1}{\sum_{\tilde{\omega}_{t,t^*+1} \in \tilde{\Omega}_{t,t^*+1}(\tilde{S}^a_{tt^*})} p(\tilde{\omega}_{t,t^*+1})} \cdot$
            $E_g[p(\widetilde{W}_{t,t^*+1})/g(\widetilde{W}_{t,t^*+1})\widetilde{V}_{tt^*}(S^{M,a}(\tilde{S}^a_{tt^*}, \widetilde{W}_{t,t^*+1}))]$
        **Step 1.5** $\tilde{S}_{tt^*} \leftarrow$ predecessor of $\tilde{S}^a_{tt^*}$
        **Step 1.6** $\Delta \leftarrow \tilde{C}(\tilde{S}_{tt^*}, \tilde{a}_{tt^*}) + \widetilde{V}^a_{tt^*}(\tilde{S}^a_{tt^*})$
        **Step 1.7** $\widetilde{V}_{tt^*}(\tilde{S}_{tt^*}) \leftarrow \widetilde{V}_{tt^*}(\tilde{S}_{tt^*}) + \frac{\Delta - \widetilde{V}_{tt^*}(\tilde{S}_{tt^*})}{N(\tilde{S}_{tt^*})}$
        **Step 1.8** $t' \leftarrow t^*$
**end while**
```

Figure 20.9 Backup process which updates the value of each decision node in the tree.

4) Backpropagation After simulating forward using our rollout policy to obtain an initial estimate of the value of being in our newly generated state, we now backtrack and obtain updated estimates of the value of each of the states on the path to the newly generated state (see figure 20.5(d) and the algorithm in figure 20.9).

Figure 20.10 shows a tree produced by an MCTS algorithm, which illustrates the varying degrees to which MCTS explores the tree. An indication that MCTS is adding value is the presence of narrow sections of the tree which are explored at much greater depth than other portions of the tree. If the tree is fairly balanced, then it means that MCTS is not pruning decisions which means it is basically enumerating the tree. Of course, the real question is how well the resulting tree works as a policy to solve the base model.

20.4 DETERMINISTIC LOOKAHEAD POLICIES WITH VECTOR DECISIONS

There is a distinct transition when we move from discrete actions $a \in \mathcal{A}$ to vectors $x \in \mathcal{X}$. While we can handle surprisingly complex stochastics and high dimensional state variables using Monte Carlo tree search, we cannot handle vector-valued decisions, since there are steps in the algorithm where we have to enumerate every single action.

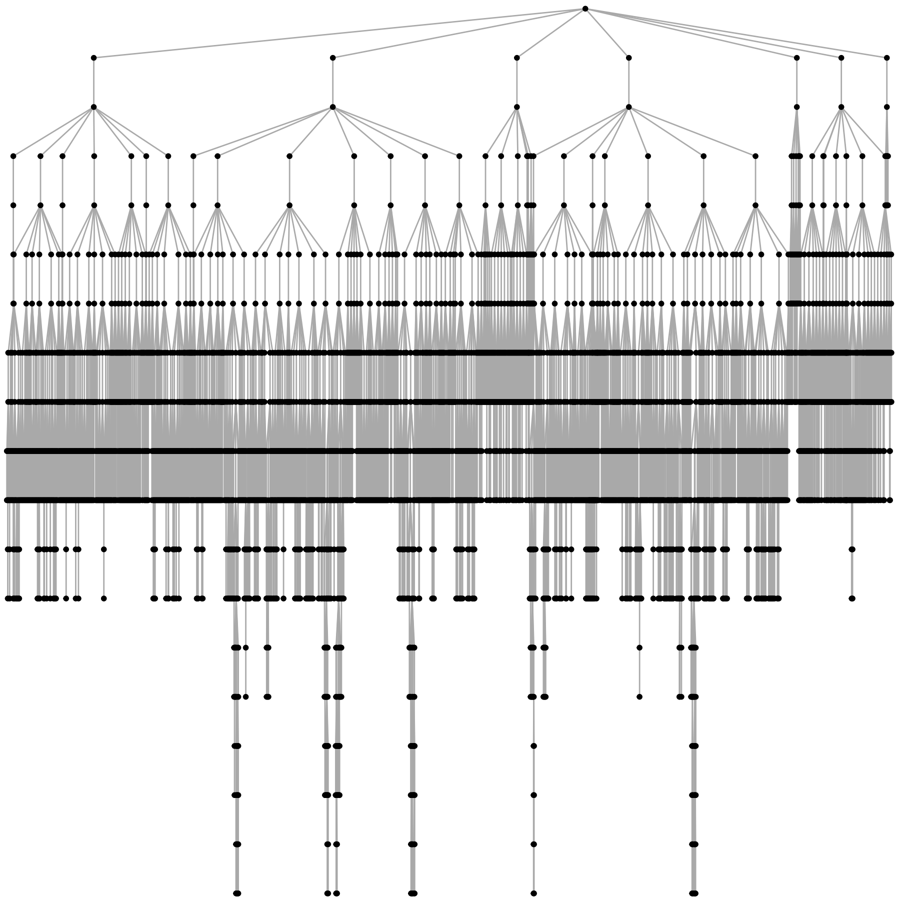


Figure 20.10 Sample of a tree produced by Monte Carlo tree search, illustrating the variable depth produced by an MCTS algorithm.

It is perhaps not surprising that the most widely used policy used in industry when decisions x_t are vector valued is to use a deterministic forecast of the future. Using our notation, we typically distinguish between the decision x_t that is to be implemented now (at time t), and decisions $\tilde{x}_{tt'}$ that we plan for a future time period t' but which only serve the purpose of helping us do a better job of determining x_t (we can think of x_t as being the same as $\tilde{x}_{tt}$).

$$X_t^{LA-Det}(S_t) = \arg\min_{x_t} \left(c_t x_t + \min_{\tilde{x}_{t,t+1},\ldots,\tilde{x}_{t,t+H}} \sum_{t'=t+1}^{t+H} \tilde{c}_{tt'} \tilde{x}_{tt'} \right), \quad (20.13)$$

subject to constraints on the current decision x_t, and the decisions in the lookahead model $(\tilde{x}_{tt'})_{t'>t}$.

t

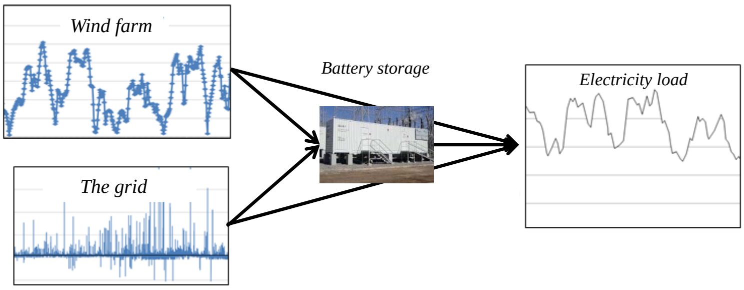


Figure 20.11 Energy storage system, with energy from a wind farm with random supply, the grid with random prices, serving a load with time-dependent demands.

20.4.1 An energy application

A popular method in practice for building a policy for stochastic, dynamic problems is to use a point forecast of future exogenous information to create a deterministic model over a H-period horizon. In fact, the term rolling horizon procedure (or model predictive control) is often interpreted to specifically refer to the use of deterministic forecasts. We illustrate the idea using a simple example drawn from energy systems analysis which is illustrated in figure 20.11. We switch to vector-valued decisions x, since a major feature of deterministic forecasts is that it makes it possible to use standard math programming solvers which scale to large problems.

Consider the problem of managing how much energy we should store in a battery to help power a building which receives energy from the electric power grid (but at a random price) or solar panels (but with random production due to cloud cover). We assume that we can always purchase power from the grid, but the price may be quite high.

Let

$$\begin{aligned}R_t &= \text{The amount of energy stored in the battery at time } t,\\ p_t &= \text{The price of electricity purchased at time } t \text{ from the grid,}\\ q_t &= \text{The energy production from the solar panel at time } t,\\ D_t &= \text{The demand for electrical power in the building at time } t.\end{aligned}$$

The state of our system is given by $S_t = (R_t, p_t, q_t, D_t)$. The system is controlled using

x_t^{gb} = The amount of energy stored in the battery from the grid at price p_t at time t,

x_t^{sb} = The amount of energy stored in the battery from the solar panels at time t,

x_t^{sd} = The amount of energy directed from the solar panels to serve demand at time t,

x_t^{bd} = The amount of energy drawn from the battery to serve demand at time t,

x_t^{gd} = The amount of energy drawn from the grid to serve demand at time t.

We let $x_t = (x_t^{gb}, x_t^{sb}, x_t^{sd}, x_t^{bd})$ be the decision vector. The cost to meet demand at time t is given by

$$C(S_t, x_t) = p_t\big(x_t^{gb} + x_t^{gd}\big).$$

The challenge with deciding what to do right now is that we have to think about demands, prices and solar production in the future. Demand and solar production tend to follow a daily pattern, although demand rises early in the morning more quickly than solar production, and can remain high in the evening after solar production has disappeared. Prices tend to be highest in the middle of the afternoon, and as a result we try to have energy stored in the battery to reduce our demand for expensive electricity at this time.

We can make the decision x_t by optimizing over the horizon from t to $t + H$. While $p_{t'}, q_{t'}$ and $D_{t'}$, for $t' > t$, are all random variables, we are going to replace them with forecasts $\bar{p}_{tt'}, \bar{q}_{tt'}$ and $\bar{D}_{tt'}$, all made with information available at time t. Since these are deterministic, we can formulate the following deterministic optimization problem:

$$\min_{\tilde{x}_{tt},\ldots,\tilde{x}_{t,t+H}} \sum_{t'=t}^{t+H} \bar{p}_{tt'}\big(\tilde{x}_{tt'}^{gb} + \tilde{x}_{tt'}^{gd}\big) - \theta^{pen}\tilde{x}_{tt'}^{slack}, \tag{20.14}$$

subject to, for $t' = t+1, \ldots, t+H$:

$$\begin{aligned}
R_{t'+1} - \big(\tilde{x}_{tt'}^{gb} + \tilde{x}_{tt'}^{sb} - \tilde{x}_{tt'}^{bd}\big) &= R_{t'}, && (20.15)\\
\tilde{x}_{tt'}^{sd} + \tilde{x}_{tt'}^{sb} &\leq \bar{q}_{tt'}, && (20.16)\\
\tilde{x}_{tt'}^{bd} + \tilde{x}_{tt'}^{sd} + \tilde{x}_{tt'}^{gd} + \tilde{x}_{tt'}^{slack} &\leq \bar{D}_{tt'}, && (20.17)\\
\tilde{x}_{tt'}^{gb}, \tilde{x}_{tt'}^{sb}, \tilde{x}_{tt'}^{bd}, \tilde{x}_{tt'}^{sd}, \tilde{x}_{tt'}^{slack} &\geq 0. && (20.18)
\end{aligned}$$

We note that we are letting $\tilde{x}_{tt'}$ be a plan of what we think we will do at time t' when we solve the optimization problem at time t. It is useful to think of this as a forecast of a decision. We project decisions over this horizon because we need to know what we would do in the future in order to know what we should do right now (which is the case with all lookahead models).

The optimization problem (20.14) - (20.18) is a deterministic linear program. We can solve this using, say, 1 minute increments over the next 12 hours (720 time periods) without difficulty. However, we are not interested in the values of $\tilde{x}_{t,t+1}, \ldots, \tilde{x}_{t,t+H}$. We are only interested in $x_t = \tilde{x}_{tt}$, which we implement at time t. As we advance the clock from t to $t+1$, we are likely to find that the random variables have not evolved precisely according to the forecast, giving us a problem starting at time $t+1$ (extending through $t+H+1$) that is slightly different than what we thought would be the case.

Our deterministic model offers several advantages. First, we have no difficulty handling the property that x_t is a continuous vector. Second, the model easily handles the highly nonstationary nature of this problem, with daily cycles in demand, prices and solar production. Third, if a weather forecast tells us that the solar production will be less than normal six hours from now, we have no difficulty taking this into account. Thus, knowledge about the future does not complicate the problem.

At the same time, the inability of the model to handle uncertainty in the future introduces significant weaknesses. One problem is that we would not feel that we need to store energy in the battery in the event that solar production *might* be lower than we expect. Second, we may wish to store electricity in the battery during periods where electricity prices are

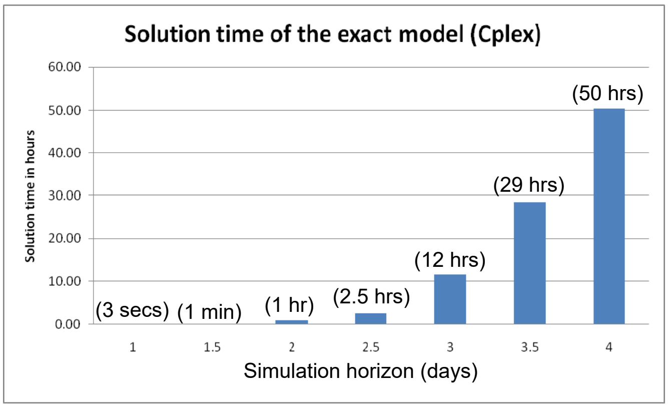


Figure 20.12 Growth in CPU times as we increase the horizon in a deterministic lookahead model.

lower than normal, something that would be ignored in a forecast of future prices, since we would not forecast stochastic variations from the mean.

Deterministic rolling horizon policies are widely used in operations research, where a deterministic approximation provides value. The real value of this strategy is that it opens the door for using commercial solvers that can handle vector-valued decisions. These policies are rarely seen in the classical examples of reinforcement learning which focus on small action spaces, but which also focus on problems where a deterministic approximation of the future would not produce an interesting model.

We close by noting that solving deterministic lookahead models, while easier than a stochastic lookahead, can actually be quite hard to solve in some situations. Figure 20.12 shows the increase in CPU times for a transportation application that involves managing locomotives over time. If we increase the horizon to four days, which is not unreasonable in this setting, the CPU time to solve a single instance of a lookahead model grows to 50 hours (this work was done around year 2010 using Cplex and a large memory computer). Similarly, there are real-time control problems (such as managing a battery controller with updates every 2 seconds), where a lookahead policy would simply be too slow.

20.4.2 A lookahead-cost function approximation

Deterministic lookahead models are widely used in practice, and widely criticized by the academic research community because the models are "deterministic." Often lost in this discussion is that the lookahead model is just a form of policy for a stochastic base model. In real applications, these are almost always modified so that the solutions work better under uncertainty. Tuning these models is stochastic optimization.

We first saw a simple example of a hybrid lookahead with modified constraints, which is a form of cost function approximation, in section 11.8 when we demonstrated that each of five classes of policies (our four core classes plus a hybrid) might work best for a simple energy storage problem. Hybrid lookahead-CFAs are a particularly powerful approach, because they combine the features of a deterministic lookahead (not too hard to solve, and can handle a forecast) with robustness against uncertainty.

We revisit the idea of a hybrid lookahead-CFA using the energy problem we introduced in section 20.4.1. We formulated a deterministic lookahead model in equations (20.14)-

(20.18), which depends on point forecasts for prices $\bar{p}_{tt'}$, solar, $\bar{q}_{tt}$, and the demand (load) $\bar{D}_{tt'}$. One way to handle the uncertainty in these forecasts is to factor them based on how far into the future we are forecasting. Instead, we can adjusts the forecasts by factors θ_τ where $\tau = t' - t$ which is how far into the future we are forecasting. Using factors for each forecast, equations (20.16)-(20.17) would become

$$\begin{aligned} \tilde{x}^{sd}_{tt'} + \tilde{x}^{sb}_{tt'} &\leq \theta^{solar}_{t'-t}\bar{q}_{tt'}, && (20.19) \\ \tilde{x}^{bd}_{tt'} + \tilde{x}^{sd}_{tt'} + \tilde{x}^{gd}_{tt'} + \tilde{x}^{slack}_{tt'} &= \theta^{load}_{t'-t}\bar{D}_{tt'}. && (20.20) \end{aligned}$$

We might use the factors to underestimate the energy from solar, or overestimate the load, to produce a conservative plan. Further, the degree of this adjustment should probably depend on how far into the future we are looking, hence the depence on $t' - t$.

Finally, we might want to have a plan where the energy in storage stays between upper and lower bounds in the future (of course, we always want to be able to use the full range of the battery at time t). Thus, we might introduce a new constraint

$$\begin{aligned} \tilde{R}_{tt'} &\leq \theta^{max}_{t'-t}R^{max}, && (20.21) \\ \tilde{R}_{tt'} &\geq \theta^{min}_{t'-t}R^{max}. && (20.22) \end{aligned}$$

We how have a set of tunable parameters $\theta = (\theta^{solar}_\tau, \theta^{load}_\tau, \theta^{min}_\tau, \theta^{max}_\tau)$ for $\tau = 1, 2, \ldots, H$. This would be called a lookup table representation because there is a different parameter θ_τ for each value of τ giving the number of time periods into the future.

An alternative strategy would be to parameterize the relationship. We might instead using a function such as

$$\theta_\tau = \alpha(1 - e^{-\beta\tau}),$$

if we want an increasing function, or

$$\theta_\tau = \alpha e^{-\beta\tau},$$

if we want a decreasing one. We would use different values of (α, β) for each type of constraint. Now, instead of having H parameters for each type of constraint, we have two.

Our policy would be written

$$X^{LA-CFA}(S_t|\theta) = \underset{\tilde{x}_{tt},\ldots,\tilde{x}_{t,t+H}}{\arg\min} \sum_{t'=t}^{t+H} \bar{p}_{tt'}\left(\tilde{x}^{gb}_{tt'} + \tilde{x}^{gd}_{tt'}\right) - \theta^{pen}\tilde{x}^{slack}_{tt'}, \qquad (20.23)$$

subject to (20.15) (which updates the inventory), along with our revised equations (20.19)-(20.22) (which is where the tunable parameters appear).

We still face the problem of finding the best value of θ. For this purpose, we return to our familiar objective function (remember that this is a minimization problem):

$$\min_\theta F^{LA-CFA}(\theta) = \mathbb{E}\sum_{t=0}^{T} C(S_t, X^{LA-CFA}(S_t|\theta)).$$

Optimizing $F^{LA-CFA}(\theta)$ has been discussed in chapter 13 on cost function approximations, as well as chapters 5 and 7 for derivative based and derivative free stochastic optimization.

20.4.3 Rolling horizon with discounting

A common objective function in dynamic programming uses discounted costs, whether over a finite or infinite horizon. If $C(S_t, x_t)$ is the contribution earned from implementing decision x_t when we are in state S_t, our objective function might be written

$$\max_\pi \mathbb{E} \sum_{t=0}^{\infty} \gamma^t C(S_t, X^\pi(S_t))$$

where γ is a discount factor that captures, typically, the time value of money. If we choose to use a rolling horizon policy (with deterministic forecasts), our policy might be written

$$X^\pi(S_t) = \underset{x_t,\ldots,x_{t+H}}{\arg\max} \sum_{t'=t}^{t+H} \gamma^{t'-t} C(S_{t'}, x_{t'}).$$

Let's consider what a discount factor might look like if it only captures the time value of money. Imagine that we are solving an operational problem where each time step is one hour (there are many applications where time steps are in minutes or even seconds). If we assume a time value of money of 10 percent per year, then we would use $\gamma = 0.999989$ for our hourly discount factor. If we use a horizon of, say, 100 hours, then we might as well use $\gamma = 1$. Not surprisingly, it is common to introduce an artificial discount factor to reflect the fact that decisions made at $t' = t + 10$ should not carry the same weight as the decision x_t that we are going to implement right now. After all, $\tilde{x}_{tt'}$ for $t' > t$ is only a forecast of a decision that might be implemented.

In our presentation of rollout heuristics, we introduced an artificial discount factor λ to reduce the influence of poor decisions backward through time. In our rolling horizon model, we are actually making optimal decisions, but only for a deterministic approximation (or perhaps an approximation of a stochastic model). When we use introduce this new discount factor, our rolling horizon policy would be written

$$X^\pi(S_t|\lambda) = \underset{x_t,\ldots,x_{t+H}}{\arg\max} \sum_{t'=t}^{t+H} \gamma^{t'-t} \lambda^{t'-t} C(S_{t'}, x_{t'}).$$

In this setting, γ plays the role of the original discount factor (which may be equal to 1.0, especially for finite horizon problems), while λ is a tunable parameter. We use λ because it is exactly analogous to the use of λ in the temporal differencing (see section 17.1.3). The motivation for the use of λ in a rolling horizon setting is the recognition that we are solving a deterministic approximation over a horizon in which events are uncertain.

20.5 TWO-STAGE STOCHASTIC PROGRAMMING

After deterministic linear programming was invented, researchers quickly realized that there were uncertainties in many applications, leading to an effort (initiated by none other than George Dantzig, who invented the simplex method that started the math programming revolution) to incorporate uncertainty into a linear program.

Not surprisingly, the introduction of uncertainty into mathematical programs opened a Pandora's box of modeling and computational issues that challenges the research community today. For decades the community focused on what became known as the two-stage stochastic programming problem, which consists of "make decision, see information, make one more decision." The two-stage stochastic programming formulation remains a foundational tool for approximating fully sequential problems. Below, we introduce the basic two-stage stochastic programming problem, and then show how it can be used to build a lookahead policy for the fully sequential inventory problem we introduced above. We then show how this can be used as an approximate lookahead model for fully sequential problems.

As a historical note: the field of stochastic programming evolved in the 1950's alongside the development of the field of Markov decision processes by Richard Bellman, with stochastic programming focusing on vector-valued decisions and Markov decision processes working with discrete action spaces. These communities evolved in parallel with distinctly different notational systems and modeling frameworks.

20.5.1 The basic two-stage stochastic program

In section 4.3.2, we introduced what is known as the *two stage stochastic program* where we make an initial decision x_0 (such as where to locate warehouses), after which we see information $W_1 = W_t(\omega)$ (which might be the demands for product), and then we make a second set of decisions $x_1(\omega)$ which depend on this information (the decisions x_1 are known as the *recourse variables*).

As we did in section 4.3.2, the two-stage stochastic programming problem is written

$$\max_{x_0} \big(c_0 x_0 + \mathbb{E} Q_1(x_0, W_1)\big), \tag{20.24}$$

subject to the constraints,

$$\begin{aligned} A_0 x_0 &= b_0, && (20.25)\\ x_0 &\geq 0, && (20.26) \end{aligned}$$

The initial decisions x_0 (which determines the inventories in the warehouses) then impacts the decisions that can be made after the information (the demand) becomes known, producing the second stage problem

$$Q_1(x_0, \omega) = \max_{x_1(\omega)} c_1(\omega) x_1(\omega), \tag{20.27}$$

subject to, for all $\omega \in \Omega$,

$$\begin{aligned} A_1 x_1(\omega) &\leq B_1 x_0, && (20.28)\\ B_1 x_1(\omega) &\leq D_1(\omega), && (20.29)\\ x_1(\omega) &\geq 0. && (20.30) \end{aligned}$$

We note that while $Q_1(x_0) = \mathbb{E} Q_1(x_0, W_1)$ is a value function, the $Q(\cdot)$ notation is standard in this literature. Oddly, while x_0 is not, strictly speaking, a state variable, it deterministically determines the state at time 1. For example, we could write

$$R_1 = B_1 x_0,$$

and then write the first stage objective as

$$\max_{x_0} \big(c_0 x_0 + \mathbb{E} Q_1(R_1, W_1)\big).$$

In fact, for many applications R_1 is lower dimensional (and possibly much lower dimensional) than x_0. For example, R_1 might be a vector of inventories, where R_{1i} is the amount of inventory at location i, while x_0 is a high dimensional vector with elements x_{0ij}. Our version above represents standard convention in this community.

We cannot actually compute this model if Ω represents all the potential outcomes, so we have to use the idea of a sampled model that we first introduced in section 4.3. We note that even when Ω is carefully designed and "not too big," the two-stage problem can still be hard to solve (if the decision vectors are large enough).

There are several computational strategies that have been used to solve problem (20.24) - (20.30):

The "deterministic equivalent" method The problem (20.24) - (20.30), when formulated using a sampled set of observations Ω, is basically a single (potentially large) deterministic linear program, leading some to refer to this problem as the "deterministic equivalent." Modern solvers can handle problems with hundreds of thousands of variables with minimal training (specialists have worked on problems with millions of variables).

Relaxation If we replace x_0 with $x_0(\omega)$, this means that we are allowing x_0 to see the future. We can fix this with a *nonanticipativity constraint* that looks like

$$x_0(\omega) = x_0, \text{ for all } \omega \in \hat{\Omega}. \tag{20.31}$$

This formulation allows us to design algorithms that relax this constraint, allowing us to solve $|\Omega|$ independent problems with logic that penalizes deviations of (20.31). This methodology has become known under the name of *progressive hedging* which has made it possible to approach two-stage stochastic programs that would otherwise be too large.

Benders decomposition In section 4.3.2 we introduced the idea of using Benders decomposition, where the function $\mathbb{E}Q_1(x_0, W_1)$ is replaced with a series of cuts. This is really a form of approximate dynamic programming, which we showed in chapter 19.

We are now going to transition to using this two-stage model as a policy for fully sequential problems.

20.5.2 Two-stage approximation of a sequential problem

While there are true two-stage stochastic programming problems, there is a vast range of fully sequential problems of the types that we have been pursuing in this book that involve vector-valued decisions in the presence of uncertainty. A widely used strategy is to solve these problems by approximating them as two-stage problems, where there is a decision to be made now at time t, represented by x_t, after which we then pretend that we observe all the future information over the rest of our horizon $(t+1, t+H)$. After this information is revealed, we then make all remaining decisions $\tilde{x}_{tt'}$ for $t' = t+1, \ldots, t+H$, which represents (along with the revealed information) the second stage of our decision problem.

In our two-stage approximation, the decisions $\tilde{x}_{tt'}$ are allowed to "see" into the future, which means that they depend on the future information. We create a sampled set of observations that we call $\tilde{\Omega}_t$ (indexed by t because it is generated at time t), where each $\tilde{\omega}_t \in \tilde{\Omega}_t$ represents a full sequence of the random variables $W_{t+1}, W_{t+2}, \ldots, W_{t+H}$.

Given $\tilde{\omega}_t \in \tilde{\Omega}_t$ (that is, given the rest of the future), we index all future decisions with the same outcome (scenario in the language of stochastic programming) $\tilde{\omega}_t$, giving us the vector $\tilde{x}_{tt'}(\tilde{\omega}_t)$, for $t' = t+1, t+2, \ldots, t+H$. These decisions violate nonanticipativity, because each $\tilde{\omega}_t \in \tilde{\Omega}_t$ represents a full realization of information over the entire planning horizon. This introduces errors, but there is a tremendous computational benefit over full multistage models (which we discuss below).

If we fix $\tilde{\omega}_t \in \tilde{\Omega}_t$, then this is like solving the deterministic optimization problem above, but instead of using forecasts of prices, $\bar{p}_{tt'}$, the energy from solar $\bar{q}_{tt'}$, and the demand $\bar{D}_{tt'}$, we use sample realizations $\tilde{p}_{tt'}(\tilde{\omega}_t)$, $\tilde{h}_{tt'}(\tilde{\omega}_t)$ and $\tilde{D}_{tt'}(\tilde{\omega}_t)$. Let $p(\tilde{\omega}_t)$ be the probability that $\tilde{\omega}_t$ happens, where this might be as simple as $p(\tilde{\omega}_t) = 1/|\tilde{\Omega}_t|$.

There are two ways to model data and decisions at time t. Consider the price p_t at time t, which is stochastic in the future. We can write $\tilde{p}_{tt'}(\tilde{\omega}_t)$ to represent a potential value of the price at time t' when we are solving a lookahead model at time t, where ω simply identifies which price in a sampled set $\tilde{\Omega}_t$. We could write the price at time t as $\tilde{p}_{tt}(\tilde{\omega}_t)$, but the price at time t is deterministic, so we could write $\tilde{p}_{tt}(\tilde{\omega}_t)$ but recognize that $\tilde{p}_{tt}(\tilde{\omega}_t) = p_t$ for all $\tilde{\omega}_t$.

Writing p_t as $\tilde{p}_{tt}(\tilde{\omega}_t)$ seems clumsy, but it takes on a different meaning when we model decisions. We need to make a single decision x_t, but we can also write $\tilde{x}_{tt}(\tilde{\omega}_t)$. However, $\tilde{\omega}_t$ is the information that becomes available in the future, while $\tilde{x}_{tt}$ is a decision that we have to make now before we know this future information. Writing $\tilde{x}_{tt}(\tilde{\omega}_t)$ means making a decision at time t knowing the future $\tilde{\omega}_t$. For this reason, we have to impose a constraint

$$\tilde{x}_{tt}(\tilde{\omega}_t) = x_t. \tag{20.32}$$

Equation (20.32) is known in the stochastic programming literature as a *nonanticipativity constraint*. The first question that a reader should ask is: why would we even use the notation $x_{tt}(\omega)$? The reason is computational. Imagine that we write the time t decision as $x_{tt}(\omega)$ and temporarily ignore equation (20.32). In this case, the problem would decompose into a series of problems, one for each $\tilde{\omega}_t \in \tilde{\Omega}_t$. These are much smaller problems than a single problem where we have to deal with all the different scenarios at the same time, but it means that we can get a different answer $\tilde{x}_{tt}(\tilde{\omega}_t)$ at time t, which is a problem. However, there are algorithmic strategies that take advantage of this problem structure.

We can model the first period decision as $\tilde{x}_{tt}(\tilde{\omega}_t)$ and impose a nonanticipativity constraint (20.32), or we can simply use x_t and just let all the future decisions $\tilde{x}_{tt'}(\tilde{\omega}_t)$ for $t' > t$ depend on the scenario. If we use the latter formulation (which is simpler to write, but may be harder to solve), we get

$$\min_{x_t, (\tilde{x}_{tt'}(\tilde{\omega}_t))_{t'=t+1}^{t+H},\, \tilde{\omega}_t \in \tilde{\Omega}_t} p_t\big(x_t^{gb} + p_t x_t^{gd}\big) + \sum_{\tilde{\omega}_t \in \tilde{\Omega}_t} P(\tilde{\omega}_t) \sum_{t'=t+1}^{t+H} \big(\tilde{p}_{tt'}(\tilde{\omega}_t)\big(\tilde{x}_{tt'}^{gb}(\tilde{\omega}_t) + \tilde{x}_{tt'}^{gd}(\tilde{\omega}_t)\big) \\ - \theta^{pen}\tilde{x}_{tt'}^{slack}\big)$$

subject to the constraints

$$\tilde{R}_{t'+1}(\tilde{\omega}_t) - \big(\tilde{x}_{tt'}^{gb}(\tilde{\omega}_t) + \tilde{x}_{tt'}^{sb}(\tilde{\omega}_t) - \tilde{x}_{tt'}^{bd}(\tilde{\omega}_t)\big) = \tilde{R}_{t'}(\tilde{\omega}_t), \tag{20.33}$$

$$\tilde{x}_{tt'}^{sd}(\tilde{\omega}_t) + \tilde{x}_{tt'}^{sb}(\tilde{\omega}_t) \quad \leq \quad \tilde{h}_{t'}(\tilde{\omega}_t), \tag{20.34}$$

$$\tilde{x}_{tt'}^{bd}(\tilde{\omega}_t) + \tilde{x}_{tt'}^{gd}(\tilde{\omega}_t) + \tilde{x}_{tt'}^{sd}(\tilde{\omega}_t) + \tilde{x}_{tt'}^{slack} = \tilde{D}_{t'}(\tilde{\omega}_t), \tag{20.35}$$

$$\tilde{x}_{tt'}^{gb}(\tilde{\omega}_t), \tilde{x}_{tt'}^{sb}(\tilde{\omega}_t), \tilde{x}_{tt'}^{bd}(\tilde{\omega}_t), \tilde{x}_{tt'}^{sd}(\tilde{\omega}_t), \tilde{x}_{tt'}^{slack} \geq 0. \tag{20.36}$$

Equation (20.33) is the flow conservation constraint for our battery, while equation (20.34) limits the power available from the solar farm. Equation (20.35) limits how much we can deliver to the customer, where $\tilde{x}^{slack}_{tt'}$ is the slack variable that captures how much demand is not satisfied, which carries a penalty θ^{pen} in the objective function.

This formulation allows us to make a decision at time t while modeling the stochastic variability in future time periods. In the stochastic programming community, the outcomes $\tilde{\omega}_t$ are often referred to as *scenarios*. If we model 20 scenarios, then our optimization problem becomes roughly 20 times larger, so the introduction of uncertainty in the future comes at a significant computational cost. However, this formulation allows us to capture the variability in future outcomes, which can be a significant advantage over using simple forecasts. Furthermore, while the problem is certainly much larger, we can approach it using one of the three algorithmic strategies described in section 20.5.1.

20.5.3 Decomposition strategies

Although the two-stage approximation is dramatically smaller than a full multistage model (as we show below), even two-stage approximations can be quite challenging. The problem is that there are many applications where even a deterministic lookahead model can be quite hard.

20.6 MULTISTAGE STOCHASTIC PROGRAMMING

Introducing uncertainty in multistage optimization problems in the presence of vector-valued decisions is intrinsically difficult, and not surprisingly there does not exist a computationally tractable algorithm to provide an exact solution to this problem. However, some practical approximations have evolved. We illustrate the simplest strategy here.

20.6.1 Modeling multistage stochastic programs

It is common to model a stochastic program as if it starts at time 0, and extends over a horizon $t = (1, 2, \ldots, T)$, ignoring the fact that this is a lookahead model that starts at time t, and extends over a horizon $t' = (t, t+1, \ldots, t+H)$. This ignores the fact that the stochastic program, which is, by itself, a challenging stochastic optimization problem, is really just a lookahead policy for another stochastic optimization problem that we have been calling the base model. We are going to stay with our notation to model the fact that our lookahead model is being created at time t in the base model, and we use t' to indicate the time within the lookahead model. We also use tilde's (most of the time) to indicate variables in the lookahead model.

We begin by describing modeling assumptions used when formulating multistage (or even two-stage) stochastic programs. Most important is to separate the physical process controlled by $\tilde{x}_{tt'}$ which determines the physical state $\tilde{R}_{tt'}$, and the information process $\tilde{I}_{tt'}$ that evolves exogenously. Combined these make up the state

$$\tilde{S}_{tt'} = (\tilde{R}_{tt'}, \tilde{I}_{tt'}).$$

These are treated separately, which also means that decisions cannot have an impact on information, an assumption that we do not require in any of our other classes of policies (PFAs, CFAs, VFAs). It is important to recognize that just as $\tilde{R}_{tt'}$ is the physical state at time t', $\tilde{I}_{tt'}$ is only the information we need at time t' to model the lookahead model from time t' onward. Thus, while $\tilde{I}_{tt'}$ might include the entire history $h_{tt'}$, in most applications $\tilde{I}_{tt'}$ is likely to be much more compact than $h_{tt'}$ (but it may still be quite high dimensional).

To model the physical process, we let $\tilde{R}_{tt'}t$ be a vector of resources, where an element might be $\tilde{R}_{t,t'k}$ where k is a location (the number of freight containers at port or rail yard k), or a type of blood k. We might also use $\tilde{R}_{tt'r}$ where $r = (r_1, \ldots, r_M)$ is a vector of attributes, perhaps to describe a driver or complex equipment such as an aircraft. The problem that arises when r is a vector is that the dimensionality of $\tilde{R}_{tt'}$ becomes extremely large. For this reason, we will assume that $\tilde{R}_{tt'} = (\tilde{R}_{tt'k})_{k \in \mathcal{K}}$ where the size of the set $\mathcal{K}$ is "not too large" (100's, perhaps 1,000's, maybe 10,000).

We then assume that our decision $\tilde{x}_{tt'}$ at time t' in the lookahead model is subject to constraints of the form

$$\tilde{A}_{tt'}\tilde{x}_{tt'} = \tilde{R}_{tt'},$$

where these are typically flow conservation constraints. We then assume that this vector evolves over time according to

$$\tilde{R}_{t,t'+1} = \tilde{B}_{tt'}\tilde{x}_{tt'} + \delta\tilde{R}_{t,t'+1}, \tag{20.37}$$

$$\tilde{A}_{t,t'+1}\tilde{x}_{t,t'+1} = \tilde{R}_{t,t'+1}, \tag{20.38}$$

where $\delta\tilde{R}_{t+1}$ represents exogenous changes (new arrivals, departures, theft of product, rainfall). Previously, we used $\hat{R}_{tt'}$ to capture these exogenous changes, but in this section, we are modeling every variable indexed by t' as if it first becomes known at time t'.

Separate from the physical process is the information process. The data in our linear program at time t' in the lookahead model consists of costs $\tilde{c}_{tt'}$, the constraint matrices $\tilde{A}_{tt'}$ and $\tilde{B}_{tt'}$, and the exogenous changes in supplies $\delta\tilde{R}_{tt'}$ (that enters the problem in $R_{t,t'+1}$ at time $t'+1$). We let $\widetilde{W}_{tt'} = (\tilde{A}_{tt'}, \tilde{B}_{tt'}, \tilde{c}_{tt'}, \delta\tilde{R}_{tt'})$ be our exogenous information process if everything is random (there are many applications where only $\delta\tilde{R}_{tt'}$ is random, while the other variables are time-dependent but deterministic).

Let $\tilde{h}_{tt'}$ be the history of our information process which we can write

$$\begin{aligned}\tilde{h}_{tt'} &= (\widetilde{W}_{tt}, \widetilde{W}_{t,t+1}, \ldots, \widetilde{W}_{tt'}),\\ &= ((\tilde{A}_{tt}, \tilde{B}_{tt}, \tilde{c}_{tt}, \delta\tilde{R}_{tt}), \ldots, (\tilde{A}_{tt'}, \tilde{B}_{tt'}, \tilde{c}_{tt'}, \delta\tilde{R}_{tt'})).\end{aligned}$$

We next let $\tilde{\Omega}_t$ be the set of all the sample paths $\widetilde{W}_{tt}, \ldots, \widetilde{W}_{t,t+H}$ over our horizon. This means that when we use an index $\tilde{\omega}_t \in \tilde{\Omega}_t$, it refers to the information over the entire planning horizon (in our lookahead model). Specifying $\tilde{\omega}_t$ is like specifying the entire future (within the lookahead model).

It is useful to be able to label all the elements $\tilde{\omega}_t$ that correspond to a particular history $\tilde{h}_{tt'}$. For this purpose, we define the set of outcomes that share a history, which we write as

$$\mathcal{H}_t(\tilde{h}_{tt'}) = \{\tilde{\omega}_t \in \tilde{\Omega}_t | (\widetilde{W}_{tt}, \ldots, \widetilde{W}_{tt'}) = h_{tt'}\}.$$

Figure 20.13 depicts a set of sample paths $\tilde{\omega}_t$ from a set $\tilde{\Omega}_t$, where we have constructed these sample paths as they might happen in reality, with branching occurring at each time period. Figure 20.14 illustrates a subset of outcomes $\mathcal{H}_t(\tilde{h}_{tt'}) \in \tilde{\Omega}_t$ where the information from t to t' matches a particular history $\tilde{h}_{tt'}$.

The set $\tilde{\Omega}_t$, when generated in a way that it captures the branching process of real information processes, is known in the stochastic programming literature as a *scenario tree*. The challenge, that we address below, is that we need to make decisions $\tilde{x}_{tt'}$ that reflect the information available at time t'. Thus, it is common to write $\tilde{x}_{tt'}(\tilde{\omega}_t)$, when in

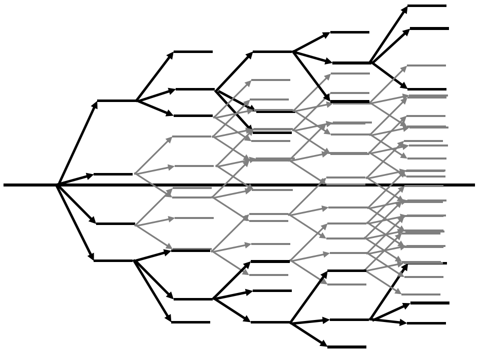


Figure 20.13 A multistage scenario tree.

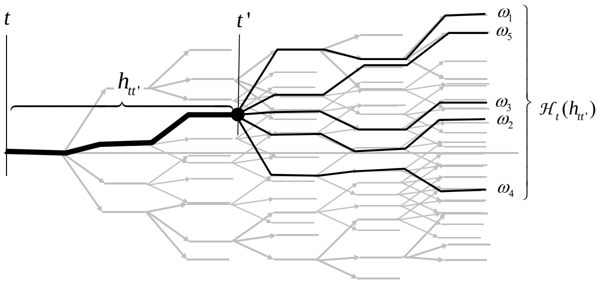


Figure 20.14 The sample paths corresponding to a history $h_{tt'}$.

fact what we want to write is $\tilde{x}_{tt'}(h_{tt'})$ to reflect the dependence of the decision on the history rather than the entire sample path.

In reality, for most applications, the decision does not even depend on the entire sample path. Rather, it only depends on what we need to know, which is what we call the state variable $\tilde{S}_{tt'}$. For example, our information process might be modeling a stochastic prices process or the speed of wind. These processes might only depend on the current value, or perhaps the trailing two or three values. In this case the state does not capture the entire history; rather it captures only what we need to know. For example, our stochastic linear program might easily be described by the state variable

$$\begin{aligned}\tilde{S}_{tt'} &= (\tilde{R}_{tt'}, (\tilde{A}_{tt'}, \tilde{B}_{tt'}, \tilde{c}_{tt'}, \delta\tilde{R}_{tt'})) \\ &= (\tilde{R}_{tt'}, \tilde{I}_{tt'}),\end{aligned}$$

where, in the language of chapter 9, $\tilde{R}_{tt'}$ is the physical state at time t', and $\tilde{I}_{tt'}$ is the information state (which includes exogenous changes to the resource state variable). While this is quite a messy state variable, at least it does not include the entire history.

20.6.2 The multistage stochastic program

A classical approach for writing a multistage stochastic optimization problems, solved at time t over a horizon $t' = t, \ldots, t+H$ is given by

$$\begin{aligned} &\max_{\substack{\tilde{A}_{t0}\tilde{x}_{t0} = \tilde{R}_{t0} \\ \tilde{x}_{t0} \geq 0}} \tilde{c}_{t0}\tilde{x}_{t0} + \mathbb{E}\left[\max_{\substack{\tilde{B}_{t0}\tilde{x}_{t0} + \tilde{A}_{t1}\tilde{x}_{t1} = \tilde{R}_{t1} \\ \tilde{x}_{t1} \geq 0}} \tilde{c}_{t1}\tilde{x}_{t1} + \mathbb{E}\left[\cdots + \right.\right. \\ &\qquad \left.\left. \mathbb{E}\left[\max_{\substack{\tilde{B}_{t,T-1}\tilde{x}_{t,T-1} + \tilde{A}_{tT}\tilde{x}_{tT} = \tilde{R}_{tT} \\ \tilde{x}_{tT} \geq 0}} \tilde{c}_{tT}, \tilde{x}_{tT} | \mathcal{H}_t(h_{tT}) \right] \ldots | \mathcal{H}_t(h_{t2}) \right] | \mathcal{H}_t(h_{t1}) \right]. \end{aligned} \tag{20.39}$$

Equation (20.39) captures the nesting of decisions and information (represented by the expectations), which precisely mimics the decision tree we presented for discrete actions. This formulation assumes that $\tilde{x}_{tt'}$ is a vector which has to satisfy constraints of the form $\tilde{A}_{tt'}\tilde{x}_{tt'} = \tilde{R}_{tt'}$.

We remind the reader that an expectation in this setting would be over the entire set of sample paths $\tilde{\Omega}_t$ where $\omega_t \in \tilde{\Omega}_t$ represents a sample of $\widetilde{W}_{tt}, \ldots, \widetilde{W}_{t,t+H}$. A conditional expectation given $\mathcal{H}_t(h_{tt'})$ simply means an expectation over the subset of $\tilde{\Omega}_t$ that belong to the set $\mathcal{H}_t(h_{tt'})$.

We treat as potentially new information (arriving between $t' - 1$ and t') the matrices $\tilde{A}_{tt'}$ and $\tilde{B}_{tt'}$, the cost vector $\tilde{c}_{tt}$ and the exogenous changes to the resource vector $\delta\tilde{R}_{tt'}$ from which we can determine $\tilde{R}_{tt'}$ (along with the previous decision $\tilde{x}_{t,t'-1}$). Thus we would write

$$\widetilde{W}_{tt'} = (\tilde{A}_{tt'}, \tilde{B}_{tt'}, \tilde{c}_{tt'}, \delta\tilde{R}_{tt'}).$$

We note that there are many applications where the only source of randomness is $\delta\tilde{R}_{tt'}$, often in the form of random demands (and possibly supplies). However, the matrices $\tilde{A}_{tt'}$ and $\tilde{B}_{tt'}$ would represent how we model random travel times, and there are many problems where costs or prices are random.

This model illustrates information where the exogenous information (that is, $(\tilde{A}_{tt'}, \tilde{B}_{tt'}, \tilde{c}_{tt'})$) goes directly into the (information) state $\tilde{I}_{tt'}$, while other information, $\delta\tilde{R}_{tt'}$, is used in the updating of the resource state $\tilde{R}_{tt'}$.

Conditioning on a set $\mathcal{H}_t(h_{tt'})$ is the same as fixing a node in the scenario tree corresponding to the end of the history $h_{tt'}$ in figure 20.14. Some authors even replace the history $h_{tt'}$ with the index of the node corresponding to the end of the history at time t', but requires that we condition on the entire history. While there are situations where this may be appropriate (applications in finance may use this), in most applications the information state does not require the entire history. In fact, in our problem the information state is just the exogenous information, which is to say

$$\begin{aligned} \tilde{I}_{tt'} &= \widetilde{W}_{tt'} \\ &= (\tilde{A}_{tt'}, \tilde{B}_{tt'}, \tilde{c}_{tt'}, \delta\tilde{R}_{tt'}). \end{aligned}$$

We caution that problems where $\tilde{I}_{tt'} = \widetilde{W}_{tt'}$ arise fairly frequently, as in most model free settings where we can only observe the state without any knowledge of the dynamics, but this will not always be the case. An information state may easily require information from recent time periods, as would arise in any setting where the dynamics are described by a time series model.

Given this structure, we may condition all of our expectations on the information state $\tilde{I}_{tt'}$ rather than the full state $\tilde{S}_{tt'}$, in which case equation (20.39) is equivalent to

$$\begin{aligned} \max_{\tilde{x}_{tt}\in\mathcal{X}_t(S_t)} C(\tilde{S}_{tt},\tilde{x}_{tt}) + \mathbb{E}\left[\max_{\tilde{x}_{t1}\in\mathcal{X}_{t1}(\tilde{S}_{t1})} C(\tilde{S}_{t1},\tilde{x}_{t1}) + \mathbb{E}\left[\ldots\right.\right. \\ \left.\left. +\mathbb{E}\left[\max_{\tilde{x}_{tT}\in\mathcal{X}_T(\tilde{S}_{tT})} C(\tilde{S}_{tT},\tilde{x}_{tT})|\tilde{I}_{tT}\right]\ldots|\tilde{I}_{t2}\right]|\tilde{I}_{t1}\right], \end{aligned} \tag{20.40}$$

which is closer to the form we have used throughout the text. Here we have replaced conditioning on the history $h_{tt'}$ with conditioning on the information state $\tilde{I}_{tt'}$ (which by construction, contains the information needed in the history). Of course, the decisions still depend on the resource state $\tilde{R}_{tt'}$, which are the only variables that depend on prior decisions.

The biggest difference between (20.40) and the standard form we have used previously is that we are not explicitly maximizing over policies. The reason is that the policy is imbedded in the formulation as

$$\tilde{X}^{SP}_{tt'}(\tilde{S}_{tt'}) = \underset{\tilde{x}_{tt'}\in\mathcal{X}_{tt'}(\tilde{S}_{tt'})}{\arg\max}\; C(\tilde{S}_{tt'},\tilde{x}_{tt'}) + \mathbb{E}\left[\ldots|\tilde{I}_{tt'}\right]$$

We again emphasize that this is just the vector-valued version of what we did in our decision tree, with the only difference being that $\tilde{I}_{tt'}$ is a node in the scenario tree of the information process (in our decision tree, the node would have corresponded to the complete state variable).

This formulation can look quite frightening. As with almost every problem in stochastic optimization, there is a computational challenge once we create the model. We have already seen how to solve these problems using VFA-based methods (such as Benders cuts) in chapter 19, but we next describe how to use a sampled model to do a true direct lookahead policy.

20.6.3 Stochastic programming with scenario trees

The optimization problem represented by the stochastic program in equation (20.39) (or more precisely, (20.40)), is really just one very large linear program that is best visualized as a decision tree, with the exception that at each node we have a linear program that is coupled with the downstream nodes (and linear programs) that are affected by the decisions we made.

We begin by constructing our scenario tree. Unlike a typical decision tree that combines decisions and (random) information, our scenario tree is constructed purely of the exogenous information process. We start with the initial state of information $\tilde{I}_{tt}$ drawn from the state S_t in the base model. We next generate a sample $\tilde{\Omega}_{tt}(\tilde{I}_{tt})$ of observations from the random variable $\widetilde{W}_{tt}$, each of which take us to a downstream information state $\tilde{I}_{t,t+1}$. We continue this from each information state $\tilde{I}_{tt'}$ until we have populated our tree. Finally, let $\mathcal{I}$ be the set of all information states (which is the same as all the nodes in the scenario tree).

The outcomes $\tilde{\omega}_{tt} \in \tilde{\Omega}_{tt'}$ each take us to a downstream information node $\tilde{I}_{t,t+1}$, so there is a downstream information state for each element of $\tilde{\Omega}_{tt'}(\tilde{I}_{tt'})$. We assign a probability

$\tilde{p}_{tt'}(\tilde{\omega}_{tt'}|\tilde{I}_{tt'})$ for each of these outcomes, where we often have

$$\tilde{p}_{tt'}(\tilde{\omega}_{tt'}|\tilde{I}_{tt'}) = \frac{1}{|\tilde{\Omega}_{tt'}(\tilde{I}_{tt'})|}.$$

Since these trees grow exponentially, a fairly common strategy is to use a larger number of outcomes for the first branching (the set $\tilde{\Omega}_{tt}$), and then use successively smaller samples.

In the discussion that follows, we are going to be making decisions $\tilde{x}_{tt'}$ at time t' (in the lookahead model), given the information state $\tilde{I}_{tt'}$ (think of this as a linear program for each node in the scenario tree, but the linear programs are linked). For this reason, we are going to need to index all of our variables by the corresponding information state. Thus, $\tilde{x}_{tt'}(\tilde{I}_{tt'})$ represents a decision made at time t' given the information state $\tilde{I}_{tt'}$. Similarly, we will have random data such as the contributions that we might label using $\tilde{c}_{tt'}(\tilde{I}_{tt'})$. Eventually, we will simply index these variables by $i \in \mathcal{I}$, but we have to use $\tilde{I}_{tt'}$ when the timing is important.

We next have to deal with the resource states $\tilde{R}_{tt'}$ which are the variables that we actually control (with noise). When we make a decision $\tilde{x}_{tt'}$ while in information state $\tilde{I}_{tt'}$, we need to know $\tilde{R}_{tt'}$. If we stepped forward in time (as we do with our other classes of policies), we would determine $\tilde{R}_{tt'}$ before having to determine $\tilde{x}_{tt'}$, but in our multistage stochastic program, we are going to be optimizing over all the decisions $\tilde{x}_{tt'}$ for $t' = t, \ldots, t+H$ at the same time.

We overcome this problem by taking advantage of the relationship between the scenario tree (that is, the set of information nodes) and the linking of the resource states through a system of linear equations. The resource state $\tilde{R}_{tt'}(\tilde{I}_{tt'})$ depends on $\tilde{x}_{t,t'-1}(\tilde{I}_{t,t'-1})$ and the information in $\tilde{I}_{tt'}$. The decisions $\tilde{x}_{t,t'-1}(\tilde{I}_{t,t'-1})$ in turn depend on $\tilde{x}_{t,t'-2}(\tilde{I}_{t,t'-2})$ and $\tilde{I}_{t,t-1}$, and so on. If we fix $\tilde{R}_{tt'}(\tilde{I}_{tt'})$, then it is fairly straightforward to solve the linear program to determine $\tilde{x}_{tt'}(\tilde{I}_{tt'})$ (if we had a way of approximating the downstream node). But we cannot fix $\tilde{R}_{tt'}(\tilde{I}_{tt'})$, since we are not solving the problems in sequence (as we have done earlier while simulating a policy). The problem is that we are optimizing over all $\tilde{x}_{tt'}(\tilde{I}_{tt'})$ for all $t' = (t, \ldots, t+H)$, and all the information states $\tilde{I}_{tt'}$, all at the same time (in one call to our linear programming solver).

What we do instead is to recognize that all of the resource variables are linked through the updating equation

$$\tilde{R}_{t,t'+1}(\tilde{I}_{t,t'+1}) \quad = \quad \tilde{B}_{tt'}(\tilde{I}_{tt'})\tilde{x}_{tt'}(\tilde{I}_{tt'}) + \delta\tilde{R}_{t,t'+1}(\tilde{I}_{t,t'+1}). \tag{20.41}$$

Equation (20.41) exploits the fact that $\tilde{I}_{t,t+1}$ uniquely determines $\tilde{I}_{tt'}$ (which in turn determines $\tilde{B}_{tt'}(\tilde{I}_{tt'})$), and $\tilde{I}_{t,t'+1}$, which also determines the exogenous resource changes $\delta\tilde{R}_{t,t'+1}(\tilde{I}_{t,t'+1})$. Equation (20.41), then, forms what can be an extremely large set of equations linking the resource variables across the entire scenario tree, producing a single, albeit potentially quite large, deterministic optimization problem.

We can now formulate our stochastic linear program. Assuming that the contribution functions are linear

$$C(\tilde{S}_{t1}, \tilde{x}_{t1}) = \tilde{c}_{tt}(\tilde{I}_{tt'})\tilde{x}_{tt'}(\tilde{I}_{tt'}), \tag{20.42}$$

we can write our stochastic linear program as

$$\max_{x_{tt'}(i), t'=t,\ldots,t+H, i\in\mathcal{I}} \sum_{t'=t}^{t+H} \tilde{c}_{tt'}(i)\tilde{x}_{tt'}(i), \tag{20.43}$$

subject to (20.41), the constraints at time (t, t')

$$\tilde{A}_{tt'}(\tilde{I}_{tt'})x_{tt'}(\tilde{I}_{tt'}) = \tilde{R}_{tt'}(\tilde{I}_{tt'}). \tag{20.44}$$

and nonnegativity constraints for each $\tilde{x}_{tt'}(i)$ for all $t' = t, \ldots, t+H$ and $i \in \mathcal{I}$.

The optimization problem (20.43) with constraints (20.41) and (20.44) (and nonnegativity) produces what may be a very large linear program. Remembering that each $x_{tt'}(i)$ is itself a vector, this can be extremely large, and as a result a community has evolved that focuses on algorithmic strategies for these problems. As of this writing, most of this research is still directed at two-stage problems, especially when integer variables are concerned.

The approach we have used here is widely used (although our notation here is new as of this writing), but it is not the only way to model a multistage stochastic program. The other approach creates a sample $\omega \in \tilde{\Omega}_t$, which determines the entire information sequence $\widetilde{W}_{tt}, \ldots, \widetilde{W}_{t,t+H}$. We would then write $\tilde{x}_{tt'}(\omega)$, which means we are letting $\tilde{x}_{tt'}$ "see" the entire future. We handle this problem as we did for two-stage stochastic programs by imposing nonanticipativity constraints. We would do this by writing a constraint such as

$$\tilde{x}_{tt'}(h_{tt'}) = \tilde{x}_{tt'}(\tilde{\omega}_t) \text{ for all } \tilde{\omega}_t \in \mathcal{H}_t(h_{tt'})$$

Ultimately, this approach is equivalent to what we have described above, which we find is more natural.

20.7 EVALUATING LOOKAHEAD POLICIES

Regardless of the approximations made to produce a tractable lookahead model in equation (20.7), it is still an optimization problem, and possibly a fairly difficult optimization problem (especially if we use a stochastic lookahead model). It is very easy to forget that this is just a class of policy, the quality of which depends on how well we have approximated the lookahead model. For this reason, we still have to evaluate the policy, possibly to help tune any parameters used, possibly to compare it against other classes of policies, and possibly to simply evaluate whether the policy is working at an acceptable level.

We can let θ parameterize all the approximations that we made in forming an approximate lookahead model. Examples include the number of stages (a stage is a combination of a decision followed by new information), the number of scenarios (when using a sampled approximation of the random variables), and the horizon. We can then represent this dependence by writing our policy as $X_t^{LA}(S_t|\theta)$, which is a form we have also used when representing policies for policy search.

We can evaluate our parameter choices θ by simulating the policy, which we can write as

$$F^{LA}(\theta^{LA}, \omega) = \sum_{t=0}^{T} C(S_t, X^{LA}(S_t(\omega)|\theta)), \tag{20.45}$$

where $S_{t+1}(\omega) = S^M(S_t(\omega), X^{LA}(S_t(\omega)|\theta), W_{t+1}(\omega))$ represents the state transition while following sample path ω^n of the exogenous process $W_1(\omega), \ldots, W_T(\omega)$. We might do one sample path (if it is long enough), or take an average over a series of samples.

The process of simulating a lookahead policy is shown in figure 20.15, which depicts a deterministic lookahead model. The lookahead model is shown on a slanted line, that indicates projecting into the future. Then, we step forward in time where we execute

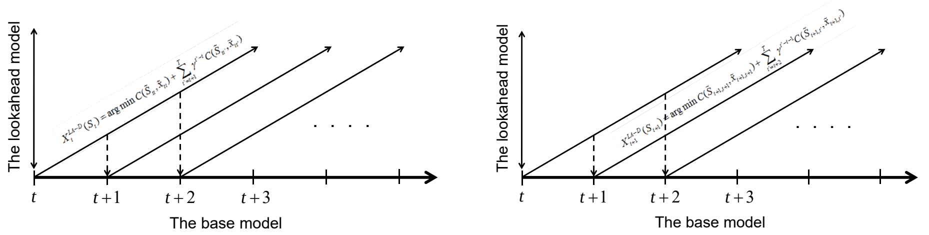

The lookahead model
$X_t^{LA-D}(S_t) = \arg\min C(\tilde{S}_{tt}, \tilde{x}_{tt}) + \sum_{t'=t+1}^{T} \gamma^{t'-t} C(\tilde{S}_{tt'}, \tilde{x}_{tt'})$
t
$t+1$
$t+2$
$t+3$
The base model
The lookahead model
$X_{t+1}^{LA-D}(S_{t+1}) = \arg\min C(\tilde{S}_{t+1,t+1}, \tilde{x}_{t+1,t+1}) + \sum_{t'=t+2}^{T} \gamma^{t'-t-1} C(\tilde{S}_{t+1,t'}, \tilde{x}_{t+1,t'})$
t
$t+1$
$t+2$
$t+3$
The base model

Figure 20.15 Illustration of rolling horizon procedure, using a deterministic model of the future (from Powell et al. (2012)).

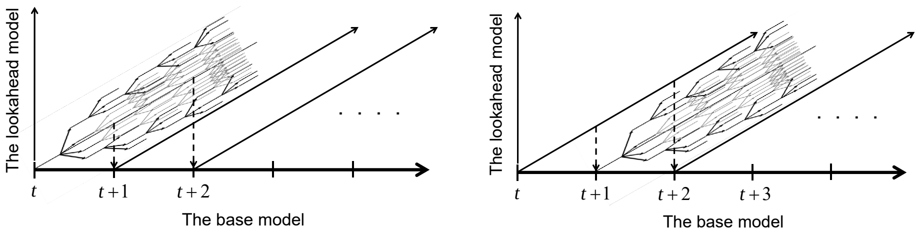


Figure 20.16 Illustration of rolling horizon procedure, using a stochastic model of the future.

$S_{t+1}(\omega) = S^M(S_t(\omega), X^{LA}(S_t(\omega)|\theta), W_{t+1}(\omega))$, which means either simulating the base model, or perhaps observing the policy work in the field.

Even if we use a stochastic lookahead based on scenario trees, we still need to run simulations to evaluate the policy. This is depicted in 20.16. Presented this way, it seems natural to simulate a stochastic lookahead policy, but this is surprisingly rare in practice, largely because stochastic lookahead policies can be quite hard to solve computationally, even with the introduction of approximations. Using backward dynamic programming for problems with state variables with three or four dimensions can require several days of computer time. Two-stage stochastic programming problems can easily require several hours (or days, depending on the implementation), and multistage stochastic programs can be virtually intractable, even when using relatively small samples. Obtaining an optimal solution to a stochastic lookahead policy is considered a major achievement, so much so that it is easy to forget that an optimal solution to an (approximate) lookahead model, even if it is stochastic, is still not an optimal policy.

Regardless of how we have approximated our lookahead model, we may need to evaluate different parameter settings which we represent by a vector θ^{LA}. This no longer makes sense when tuning a lookahead policy, since it is often the case that a lookahead parameter (such as the horizon, the number of samples in the lookahead model, the number of stages) satisfies the property of "the bigger the better." The problem is that as we make the lookahead model more realistic (longer horizons, larger samples, more stages), the lookahead model becomes much harder to solve.

We could create a utility function that trades off the performance of the policy (evaluated using (20.45)) against computational costs. In practice, scientists have a sense of their computational budget, but there is typically a nonlinear tradeoff which has to be evaluated empirically. The decision of how to choose θ is typically ad hoc, if for no other reason than the reality that most projects that involve creating a lookahead policy (especially stochastic lookaheads), do not create the type of simulator that is required to compute estimates of (20.45).

20.8 DISCUSSION

A number of comments are in order regarding the use of lookahead policies:

- It is common when using lookahead policies to form a model, and then assume that this is the model we have to solve. The real problem is the base model, while the lookahead policy is just one way of solving the base model.
- Optimal solutions of stochastic lookahead models can be quite difficult to find, but an optimal solution of an approximate lookahead model is not an optimal policy.
- It is sometimes hard to know if a model is a lookahead model or a base model. There are many instances of stochastic dynamic programs which are base models, but these might also be lookahead models. We provide some guidance below.
- While simulating a policy (any policy) is typically the best way to tune and compare policies, this is critical when using policy search (policy function approximations, or cost function approximations). By contrast, it is less important when using lookahead models. Given the complexity of building simulators, along with the approximations inherent in any simulation, there are many situations where lookahead policies are just tested in the field.

As of this writing, the vocabulary of "base models" and "lookahead models" has not entered the language of modeling in stochastic optimization. As a result, it can be difficult to identify whether a stochastic optimization model is a base model or lookahead model. Some guidelines include:

- How is the model being used? If the primary output of a model is the decision we make in the first time period, this is almost always a lookahead model. However, there are instances where the lookahead model is the same as the base model, in which case our policy is an optimal policy. On the other hand, it is possible the model is being used to answer strategic questions, where we are using optimization to simulate good decisions, then this is a base model.
- If we are using a deterministic lookahead model, or a stochastic lookahead using scenario trees, to make decisions in a setting that is stochastic, then this is a lookahead model. The future decisions in the lookahead model (that is, $\tilde{x}_{tt'}$ for $t' > t$) are not going to be implementable in a stochastic setting.

Perhaps the most important take-away of this chapter is the need to clearly distinguish the lookahead model from the base model, and to remember that the problem we are trying to solve is the base model. The lookahead model is just a form of policy, which we may compare to other policies we have discussed in this book.

20.9 BIBLIOGRAPHIC NOTES

PART VI - RISK

Up to now this book has dealt almost exclusively with expectations. This reflects both the rich literature and wide range of applications. Yet, it is easy to make the case that in the presence of uncertainty, risk is of central importance.

CHAPTER 21

RISK AND ROBUSTNESS

Utility functions
Risk measures
Dynamic risk measures
Importance sampling? Lina's work and Daniel Jiang's work

21.0.1 Risk-based and robust objective functions

xxxx

Of course, this formulation assumes that we are maximizing the expected value of the sum. Eventually (well, not until chapter 21) we are going to consider the issue of risk. For example, there are problems in energy systems where the cost of power for a building operator depends on the peak usage over the course of a month. In this case, we are trying to design an energy control policy that minimizes the maximum consumption of energy across all the time periods.

To handle these situations, we introduce the risk measure $\rho(\cdot)$ which acts on the full sequence of costs or contributions. The key difference between our risk operator $\rho(\cdot)$ and the expectation $\mathbb{E}(\cdot)$ is that the expectation of a sum is equal to the sum of the expectations, while the risk operator is nonlinear. To illustrate, let $C_t^\pi = C(S_t, X_t^\pi(S_t), W_{t+1})$ for

$t = 0, \ldots, T-1$, and $C^\pi_T = C(S_T, X^\pi_T(S_T))$ and let

$$F^\pi = \sum_{t=0}^{T} C^\pi_t,$$

where F^π is a random variable. Examples of risk operators are

$$\begin{aligned} \rho(C_0, C_1, \ldots, C_T) &= \max_t C^\pi_t, \\ \rho(C_0, C_1, \ldots, C_T) &= (F^\pi)^2. \end{aligned}$$

In this case, we write our objective as

$$\max_{\pi \in \Pi} \rho\left(C_0, C_1, \ldots, C_T\right).$$

xxx

When you introduce uncertainty, there is a number of applications in which doing well on average is not enough. There are two approaches that are being actively researched for handling these situations. The first is to replace the expectation with a risk measure that we denote by ϱ which can take many forms. One version uses a weighted combination of the mean and variance, given by

$$\varrho(X) = \mathbb{E}X + \eta\mathbb{E}\left[(X - \mathbb{E}X)^2\right]^{\frac{1}{2}},$$

where η is a parameter that controls how much weight we put on the variance. Often we wish to focus on tails, so we might use

$$\varrho(X|\alpha) = \mathbb{E}X + \eta\mathbb{E}\max\{0, X - \alpha\},$$

for a given threshold α, where outcomes of X above α are of special interest. We could turn this around, using $\max\{0, \alpha - X\}$ if our concern is with values of X less than α, as might arise if we are worried about running out of energy (or money).

A different objective that has been receiving growing attention under the label of *robust optimization* , which focuses on the worst case outcome within some constructed uncertainty set. Robust optimization originated as a static optimization problem, where you make a deterministic decision x and then observe a stochastic outcome W. A classical example involves designing a building or device that will withstand the worst possible outcome (the wind on a building, the stress on an airplane wing, the load on a transformer). We first define an *uncertainty set* $\mathcal{W}(\theta)$, which might be envisioned as 95 percent confidence intervals (if $\theta = .05$). The static robust optimization problem would be formulated as

$$\min_x \max_{w \in \mathcal{W}(\theta)} F(x, w). \tag{21.1}$$

Recently, this idea has been applied to multiperiod problems. Imagine that we are trying to make a decision x_t at time t that reflects this worst-case thinking. We can extend the idea of robust optimization for static problems by building a multiperiod uncertainty set $\mathcal{W}_{t,t+H}(\theta)$ for the random variables $W_t, \ldots, W_{t+H}$. For example, we could create 95 percent confidence intervals for each $W_{t'}$ over the entire horizon. We can then construct a robust policy by solving

$$X^\pi_t(S_t) = \arg\min_{x_t, \ldots, x_{t+H}} \max_{w_t, \ldots, w_{t+H}} \sum_{t'=t}^{t+H} c_{t'}(w_{t'}) x_{t'}. \tag{21.2}$$

This is a deterministic optimization problem which returns x_t, which is the decision that would be implemented at time t. However, equation (21.2) is not a model of a problem (that is, a base model); it is a form of lookahead policy. We revisit lookahead policies in chapter 20.

Risk measures, including the worst-case objective of robust optimization, are significantly harder to work with than expectation. Perhaps the biggest source of complexity is that risk operators, which is to say that the risk associated with the sum is not the sum of the risks (a probably that is enjoyed by expectations). This is less of a problem for policy search algorithms (discussed in chapter 12, but represents a major hurdle for algorithms that depend on Bellman's equation. The field of *dynamic risk measures* has evolved to address this issue. We address this topic in greater depth in chapter 21. ,

21.1 BIBLIOGRAPHIC NOTES

- Section xx -

Bibliography

Andreatta, G. & Romeo, L. (1988), 'Stochastic Shortest Paths with Recourse', *Networks* **18**, 193–204.

Ankenman, B., Nelson, B. L. & Staum, J. (2009), 'Stochastic Kriging for Simulation Metamodeling', *Operations Research* **58**(2), 371–382.

Auer, P., Cesa-Bianchi, N. & Fischer, P. (2002), 'Finite time analysis of the multiarmed bandit problem', *Machine Learning* **47**, 235–256.

Baird, L. C. (1995), 'Residual algorithms: Reinforcement learning with function approximation', *In Proceedings of the Twelfth International Conference on Machine Learning* **pp**, 30–37.

Barto, A. G., Sutton, R. S. & Anderson, C. W. (1983), 'Neuron-like elements that can solve difficult learning control problems', *IEEE Transactions on Systems, Man and Cybernetics* **13**(5), 834–846.

Bean, J. C., Birge, J. R. & Smith, R. L. (1987), 'Aggregation in Dynamic Programming', *Operations Research* **35**, 215–220.

Bechhofer, R. E., Santner, T. J. & Goldsman, D. M. (1995), *Design and analysis of experiments for statistical selection, screening, and multiple comparisons*, John Wiley & Sons, New York.

Bellman, R. & Kalaba, R. (1959*a*), 'On adaptive control processes', *IRE Transactions on Automatic Control* **4**(2), 1–9.

Bellman, R. E. (1957), *Dynamic Programming*, Princeton University Press, Princeton, N.J.

Bellman, R. E. (1971), *Introduction to the Mathematical Theory of Control Processes, Vol. II*, Academic Press, New York.

Bellman, R. E. & Kalaba, R. (1959*b*), 'On adaptive control processes', *OIRE Trans.* **4**, 1–9.

Berry, D. A. & Fristedt, B. (1985), *Bandit Problems*, Chapman and Hall, London.

Bertsekas, D. (2009), Approximate Dynamic Programming, *in* 'Dynamic Programming and Optimal Control 3rd Edition, Volume II', Vol. II, Athena Scientific, Belmont, MA, chapter 6.

Bertsekas, D., Borkar, V. S. & Nedic, A. (2004), Improved Temporal Difference Methods with Linear Function Approximation, *in* J. Si, A. G. Barto, W. B. Powell & D. Wunsch, eds, 'Handbook of Learning and Approximate Dynamic Programming', IEEE Press, New York, pp. 233–257.

Bertsekas, D. P. & Castanon, D. A. (1989), 'Adaptive Aggregation Methods for Infinite Horizon Dynamic Programming', *IEEE Transactions on Automatic Control* **34**, 589–598.

Bertsekas, D. P. & Castanon, D. A. (1999), 'Rollout Algorithms for Stochastic Scheduling Problems', *J. Heuristics* **5**, 89–108.

Bertsekas, D. P. & Tsitsiklis, J. N. (1996), *Neuro-Dynamic Programming*, Athena Scientific, Belmont, MA.

Bertsekas, D. P., Tsitsiklis, J. N. & An (1991), 'Analysis of stochastic shortest path problems', *Mathematics of Operations Research* **16**(3), pp580–595.

Bhatnagar, S., Sutton, R. S., Ghavamzadeh, M. & Lee, M. (2009), 'Natural actorâ^critic algorithmsâŸë', *Automatica* **45**(11), 2471–2482.

Birge, J. R. (1985), 'Decomposition and partitioning techniques for multistage stochastic linear programs', *Mathematical Programming* **33**, 25–41.

Birge, J. R. & Louveaux, F. (1997), *Introduction to Stochastic Programming*, Springer Verlag, New York.

Bishop, C. M. (2006), *Pattern Recognition and Machine Learning*, Springer, New York.

Blum, J. (1954*a*), 'Multidimensional stochastic approximation methods'', *Annals of Mathematical Statistics* **25**, 737–74462.

Blum, J. R. (1954*b*), 'Approximation Methods which Converge with Probability One', *Annals of Mathematical Statistics* **25**, 382–386.

Borkar, V. & Konda, V. (1997), 'The actor-critic algorithm as multi-time-scale stochastic approximation', *Sadhana* **22**(4), 525–543.

Boutilier, C., Dean, T. & Hanks, S. (1999), 'Decision-Theoretic Planning: Structural Assumptions and Computational Leverage', *Access* pp. 1–94.

Bradtke, S. J. & Barto, A. G. (1996), 'Linear least-squares algorithms for temporal difference learning', *Machine Learning* **22**(1), 33–57.

Brown, R. G. (1959), *Statistical Forecasting for Inventory Control*, McGraw-Hill, New York.

Brown, R. G. (1963), *Smoothing, Forecasting and Prediction of Discrete Time Series*, Prentice-Hall, Englewood Cliffs, N.J.

Bubeck, S. & Cesa-Bianchi, N. (2012), 'Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems', *Foundations and Trends® in Machine Learning* **5**(1), 1–122.

Camacho, E. & Bordons, C. (2003), *Model Predictive Control*, Springer, London.

Cant, L. (2006), 'Life Saving Decisions: A Model for Optimal Blood Inventory Management'.

Chen, C. H. (1995), An effective approach to smartly allocate computing budget for discrete event simulation, *in* '34th IEEE Conference on Decision and Control', Vol. 34, New Orleans, LA, pp. 2598–2603.

Chen, H. C., Chen, C. H., Dai, L. & Yucesan, E. (1997), A gradient approach for smartly allocating computing budget for discrete event simulation, *in* J. Charnes, D. Morrice, D. Brunner & J. Swain, eds, 'Proceedings of the 1996 Winter Simulation Conference', IEEE Press, Piscataway, NJ, USA, pp. 398–405.

Chen, H. C., Chen, C. H., Yucesan, E. & Yücesan, E. (2000), 'Computing efforts allocation for ordinal optimization and discrete event simulation', *IEEE Transactions on Automatic Control* **45**(5), 960–964.

Chen, Z.-L. L. & Powell, W. B. (1999), 'A Convergent Cutting-Plane and Partial-Sampling Algorithm for Multistage Linear Programs with Recourse', *Journal of Optimization Theory and Applications* **103**(3), 497–524.

Chick, S. E., K & Inoue, K. (2001), 'New two-stage and sequential procedures for selecting the best simulated system', *Operations Research* **49**(5), 732—-743.

Chong, E. K. P. (1991), On-Line Stochastic Optimization of Queueing Systems.

Chow, G. (1997), *Dynamic Economics*, Oxford University Press, New York.

Chung, K. L. (1974), *A Course in Probability Theory*, Academic Press, New York.

Cinlar, E. (2011), *Probability and Stochastics*, Springer, New York.

Clément, E., Lamberton, D., Protter, P. & Clement, E. (2002), 'An analysis of a least squares regression method for American option pricing', *Finance and Stochastics* **17**(4), 448–471.

Dantzig, G. B. & Ferguson, A. (1956), 'The Allocation of Aircrafts to Routes: An Example of Linear Programming Under Uncertain Demand', *Management Science* **3**, 45–73.

Darken, C. & Moody, J. (1991), Note on Learning Rate Schedules for Stochastic Optimization, *in* R. P. Lippmann, J. Moody & D. S. Touretzky, eds, 'Advances in Neural Information Processing Systems 3', pp. 1009–1016.

Darken, C., Chang, J. & Moody, J. (1992), ‘Learning Rate Schedules for Faster Stochastic Gradient Search’, *Neural Networks for Signal Processing 2 - Proceedings of the 1992 IEEE Workshop*.

de Farias, D. P. & Van Roy, B. (2003), ‘The Linear Programming Approach to Approximate Dynamic Programming’, *Operations Research* **51**, 850–865.

Dearden, R., Friedman, N. & Andre, D. (1998*a*), ‘Model based Bayesian Exploration’, *Learning*.

Dearden, R., Friedman, N. & Russell, S. (1998*b*), ‘Bayesian Q-learning’, *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE* pp. 761–768.

DeGroot, M. H. (1970), *Optimal Statistical Decisions*, John Wiley and Sons.

Denardo, E. V. (1982), *Dynamic Programming*, Prentice-Hall, Englewood Cliffs, NJ.

Douglas, S. C. & Mathews, V. J. (1995), ‘Stochastic Gradient Adaptive Step Size Algorithms for Adaptive Filtering’, *Proc. International Conference on Digital Signal Processing, Limassol, Cyprus* **1**, 142–147.

Dreyfus, S. & Law, A. M. (1977), *The Art and Theory of Dynamic Programming*, Academic Press, New York.

Dvoretzky, A. (1956), On Stochastic Approximation, *in* J. Neyman, ed., ‘Proceedings 3rd Berkeley Symposium on Mathematical Statistics and Probability’, University of California Press, pp. 39–55.

Dynkin, E. B. & Yushkevich, A. A. (1979), ‘Controlled Markov processes’, *in volume Grundlehren der mathematischen Wissenschaften 235 of A Series of Comprehensive Studies in Mathematics. New York: SpringerVerlag*.

Ermoliev, Y. (1988), Stochastic Quasigradient Methods, *in* Y. Ermoliev & R. Wets, eds, ‘\em Numerical Techniques for Stochastic Optimization’, Springer-Verlag, Berlin.

Even-dar, E. & Mansour, Y. (2003), ‘Learning rates for Q-learning’, *Journal of Machine Learning Research* **vol**, 5pp1–25.

Farias, D. & Roy, B. (2001), ‘On constraint sampling for the linear programming approach to approximate dynamic’, *Math. of Operations Res* **29**(3), 462–478.

Farias, D. P. D. & Roy, B. V. (2003), ‘The Linear Programming Approach to Approximate Dynamic Programming’, *Operations Research* **51**(6), 850 – 865.

Frank, H. (1969), ‘Shortest Paths in Probabilistic Graphs’, *Operations Research* **17**, 583–599.

Frazier, P. I., Powell, W. B. & Dayanik, S. (2009), ‘The Knowledge-Gradient Policy for Correlated Normal Beliefs’, *INFORMS Journal on Computing* **21**(4), 599–613.

Frazier, P. I., Powell, W. B. & Dayanik, S. E. (2008), ‘A knowledge gradient policy for sequential information collection’, *SIAM Journal on Control and Optimization* **47**(5), 2410–2439.

Frieze, A. & Grimmet, G. (1985), 'The Shortest Path Problem For Graphs With Random Arc Lengths', *Discrete Applied Mathematics* **10**, 57–77.

Fu, M. C., Hu, J.-Q., Chen, C.-H. & Xiong, X. (2007), 'Simulation Allocation for Determining the Best Design in the Presence of Correlated Sampling', *INFORMS J. on Computing* **19**, 101–111.

Gaivoronski, A. (1988), Stochastic quasigradient methods and their implementation, *in* Y. Ermoliev & R. Wets, eds, 'Numerical Techniques for Stochastic Optimization', Springer-Verlag, Berlin.

Gardner, E. S. (1983), 'Automatic Monitoring of Forecast Errors', *Journal of Forecasting* **2**, 1–21.

George, A., Powell, W. B. & Kulkarni, S. (2008), 'Value Function Approximation using Multiple Aggregation for Multiattribute Resource Management', *J. Machine Learning Research* pp. 2079–2111.

Giffin, W. C. (1971), *Introduction to Operations Engineering*, R. D. Irwin, Inc., Homewood, IL.

Gittins, J. (1979), 'Bandit processes and dynamic allocation indices', *Journal of the Royal Statistical Society. Series B (Methodological)* **41**(2), 148–177.

Gittins, J. (1981), 'Multiserver scheduling of jobs with increasing completion times', *Journal of Applied Probability* **16**, 321–324.

Gittins, J. (1989), 'Multi-armed Bandit Allocation Indices', *Wiley and Sons: New York*.

Gittins, J. & Jones, D. (1974), A dynamic allocation index for the sequential design of experiments, *in* J. Gani, ed., 'Progress in statistics', North Holland, Amsterdam, pp. 241—-266.

Golub, G. H. & Loan, C. F. V. (1996), *Matrix Computations*, John Hopkins University Press, Baltimore, MD.

Goodwin, G. C. & Sin, K. S. (1984), *Adaptive Filtering and Control*, Prentice-Hall, Englewood Cliffs, NJ.

Guestrin, C., Koller, D. & Parr, R. (2003), 'Efficient Solution Algorithms for Factored MDPs', *Journal of Artificial Intelligence Research* **19**, 399–468.

Gupta, S. & Miescke, K. (1996), 'Bayesian look ahead one-stage sampling allocations for selection of the best population', *Journal of statistical planning and inference* **54**(2), 229—-244.

Hastie, T. J., Tibshirani, R. J. & Friedman, J. H. (2009), *The elements of statistical learning : data mining, inference, and prediction*, Springer, New York.

Haykin, S. (1999), *Neural Networks: A comprehensive foundation*, Prentice Hall, Englewood Cliffs, N.J.

He, D., Chick, S. E. & Chen, C.-h. (2007), 'Opportunity Cost and OCBA Selection Procedures in Ordinal Optimization for a Fixed Number of Alternative Systems', *IEEE Transactions on Systems Man and Cybernetics Part C-Applications and Reviews* **37**(5), 951–961.

Heuberger, P. S. C., den Hov, P. M. J. V. & Wahlberg, B., eds (2005), *Modeling and Identification with Rational Orthogonal Basis Functions*, Springer, New York.

Heyman, D. P. & Sobel, M. (1984), *Stochastic Models in Operations Research, Volume II: Stochastic Optimization*, McGraw Hill, New York.

Higle, J. L. & Sen, S. (1991), 'Stochastic decomposition: An algorithm for two-stage linear programs with recourse', *Mathematics of Operations Research* **16**(3), 650–669.

Holt, C. C., Modigliani, F., Muth, J. & Simon, H. (1960), *Planning, Production, Inventories and Work Force*, Prentice-Hall, Englewood Cliffs, NJ.

Hong, J. & Nelson, B. L. (2006), 'Discrete Optimization via Simulation Using COMPASS', *Operations Research* **54**(1), 115–129.

Hong, L. & Nelson, B. L. (2007), 'A framework for locally convergent random-search algorithms for discrete optimization via simulation', *ACM Transactions on Modeling and Computer Simulation* **17**(4), 1–22.

Howard, R. A. (1960), Dynamic programming and Markov processes, MIT Press, Cambridge, MA.

Jaakkola, T., Jordan, M. & Singh, S. P. (1994*a*), 'On the convergence of stochastic iterative dynamic programming algorithms', *Neural Computation* **1201**(1988), 1185–1201.

Jaakkola, T., Jordan, M. I. & Singh, S. P. (1994*b*), 'On the convergence of stochastic iterative dynamic programming algorithms', *Neural Computation* **6**(6), 1185–1201.

Judd, K. L. (1998), *Numerical Methods in Economics*, MIT Press.

Kaelbling, L. P. (1993), *Learning in embedded systems*, MIT Press, Cambridge, MA.

Kall, P. & Mayer, J. (2005), *Stochastic Linear Programming: Models, Theory, and Computation*, Springer, New York.

Kesten, H. (1958), 'Accelerated Stochastic Approximation', *The Annals of Mathematical Statistics* **29**, 41–59.

Kiefer, J. & Wolfowitz, J. (1952), 'Stochastic estimation of the maximum of a regression function', *Annals of Mathematical Statistics* **23**, 462–466.

Kirk, D. E. (2004), *Optimal Control Theory: An introduction*, Dover, New York.

Kmenta, J. (1997), *Elements of Econometrics*, University of Michigan Press, Ann Arbor, MI.

Konda, V. R. & Borkar, V. S. (1999), Actor-Critic–Type Learning Algorithms for Markov Decision Processes, *in* 'SIAM Journal on Control and Optimization', Vol. 38, p. 94.

Konda, V. R. & Tsitsiklis, J. N. (2003), 'On actor-critic algorithms âš', *SIAM J. Control and Optimization* **42**(4), 1143–1166.

Kushner, H. J. & Clark, S. (1978), *Stochastic Approximation Methods for Constrained and Unconstrained Systems*, Springer-Verlag, New York.

Kushner, H. J. & Yin, G. G. (1997), *Stochastic Approximation Algorithms and Applications*, Springer-Verlag, New York.

L., l. & Soderstrom, T. (1983), *Theory and Practice of Recursive Identification*, MIT Press, Cambridge, MA.

Lagoudakis, M. & Parr, R. (2003), 'Least-squares policy iteration', *The Journal of Machine Learning Research* **4**, 1149.

Lagoudakis, M., Parr, R. & Littman, M. (2002), 'Least-squares methods in reinforcement learning for control', *Methods and Applications of Artificial Intelligence* pp. 752–752.

Lai, T. L. (1987), 'Adaptive Treatment Allocation and the Multi-Armed Bandit Problem', *Annals of Statistics* **15**(3), 1091–1114.

Lai, T. L. & Robbins, H. (1985), 'Asymptotically Efficient Adaptive Allocation Rules', *Advances in Applied Mathematics* **6**, 4–22.

Lambert, T., Smith, R. & Epelman, M. (2002), 'Aggregation in Stochastic Dynamic Programming', *Working Paper*.

LeBlanc, M. & Tibshirani, R. (1996), 'Combining estimates in regression and classification', *Journal of the American Statistical Association* **91**, 1641–1650.

Longstaff, F. A. & Schwartz, E. S. (2001), 'Valuing American options by simulation: A simple least squares approach', *The Review of Financial Studies* **14**(1), 113–147.

Ma, J. & Powell, W. B. (2010), Convergence Analysis of Kernel-based On-policy Approximate Policy Iteration Algorithms for Markov Decision Processes with Continuous , Multidimensional States and Actions.

Manne, A. S. (1960), 'Linear programming and sequential decisions', *Management Science* **6**(3), 259–267.

Marbach, P. & Tsitsiklis, J. N. (2001), 'Simulation-based optimization of Markov reward processes', *Automatic Control, IEEE Transactions on* **46**(2), 191–209.

Mayer, J. (1998), *Stochastic linear programming algorithms: A comparison based on a model management system*, Springer.

Menache, I., Mannor, S. & Shimkin, N. (2005), 'Basis function adaptation in temporal difference reinforcement learning', *Annals of Operations Research* **134**(1), 215–238.

Mendelssohn, R. (1982), 'An Iterative Aggregation Procedure for Markov Decision Processes', *Operations Research* **30**, 62–73.

Mirozahmedov, F. & Uryasev, S. (1983), 'Adaptive Stepsize Regulation for Stochastic Optimization Algorithm', *Zurnal vicisl. mat. i. mat. fiz. 23* **6**, 1314–1325.

Mulvey, J. M. & Ruszczyński, A. (1995), 'A new scenario decomposition method for large scale stochastic optimization', *Operations Research* **43**, 477–490.

Munos, R. & Szepesvári, C. (2008), 'Finite-Time Bounds for Fitted Value Iteration', *Journal of Machine Learning Research* **1**, 815–857.

Nedic, A., Bertsekas, D. P., Science, C., Nedić, A., Nediç, A. & Nedi, A. (2003), 'Least squares policy evaluation algorithms with linear function approximation', *Discrete Event Dynamic Systems* **13**(1), 79–110.

Negoescu, D. M., Frazier, P. I. & Powell, W. B. (2010), 'The Knowledge-Gradient Algorithm for Sequencing Experiments in Drug Discovery', *INFORMS Journal on Computing* pp. 1–18.

Nelson, B. L. & Kim, S.-H. (2001), 'A fully sequential procedure for indifference-zone selection in simulation', *ACM Trans. Model. Comput. Simul.* **11**(3), 251–273.

Nelson, B. L., Swann, J., Goldsman, D. & Song, W. (2001), 'Simple procedures for selecting the best simulated system when the number of alternatives is large', *Operations Research* **49**, 950–963.

Nemhauser, G. L. (1966), *Introduction to dynamic programming*, John Wiley & Sons, New York.

Ormoneit, D. & Glynn, P. W. (2002), 'Kernel-based reinforcement learning average-cost problems', *IEEE Trans. on Automatic Control* **vol**, 1624–1636.

Ormoneit, D. & Sen, Ś. (2002), 'Kernel-based reinforcement learning', *Machine Learning* **49**, 161–178.

Pflug, G. (1988), Stepsize rules, stopping times and their implementation in stochastic quasi-gradient algorithms, *in* 'Numerical Techniques for Stochastic Optimization', Springer-Verlag, New York, pp. 353–372.

Pflug, G. (1996), *Optimization of Stochastic Models: The Interface Between Simulation and Optimization*, Kluwer International Series in Engineering and Computer Science: Discrete Event Dynamic Systems, Kluwer Academic Publishers, Boston.

Pollard, D. (2002), *A User's Guide to Measure Theoretic Probability*, Cambridge University Press, Cambridge.

Porteus, E. L. (1990), Handbooks in Operations Research and Management Science: Stochastic Models, Vol. 2, North Holland, Amsterdam, chapter Stochastic, pp. 605–652.

Powell, W. B. & Cheung, R. K.-M. (2000), 'SHAPE: A Stochastic Hybrid Approximation Procedure for Two-Stage Stochastic Programs', *Operations Research* **48**, 73–79.

Powell, W. B. & George, A. P. (2006), 'Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming', *Journal of Machine Learning* **65**(1), 167–198.

Powell, W. B. & Meisel, S. (2016), 'Tutorial on Stochastic Optimization in Energy II: An energy storage illustration', *IEEE Transactions on Power Systems* **31**(2), 1459–1467.

Powell, W. B., Frazier, P. I. & Dayanik, S. (2008), 'A knowledge-gradient policy for sequential information collection', *SIAM Journal on Control and Optimization (to appear)*.

Powell, W. B., Ruszczyński, A. & Topaloglu, H. (2004), 'Learning Algorithms for Separable Approximations of Discrete Stochastic Optimization Problems', *Mathematics of Operations Research* **29**(4), 814–836.

Powell, W. B., Simao, H. P. & Bouzaiene-Ayari, B. (2012), 'Approximate dynamic programming in transportation and logistics: a unified framework', *EURO Journal on Transportation and Logistics* **1**(3), 237–284.

Powell, W. B., Simao, H. P. & Shapiro, J. A. (2001), A representational paradigm for dynamic resource transformation problems, in R, *in* F. C. Coullard & J. Owens, H., eds, 'Annals of Operations Research', J. C. Baltzer AG, pp. 231–279.

Powell, W. B., Spivey, M. Z. & Engineering, F. (2003), 'The Dynamic Assignment Problem', *Operations Research*.

Precup, D., Sutton, R. S. & Dasgupta, S. (2001), Off-policy temporal-difference learning with function approximation, *in* '19th International Conference on Machine Learning', pp. 417–424.

Psaraftis, H. N. & Tsitsiklis, J. N. (1993), 'Dynamic Shortest Paths in Acyclic Networks with Markovian Arc Costs', *Operations Research* **41**, 91–101.

Puterman, M. (2005), *Markov Decision Processes*, 2nd edn, John Wiley & Sons Inc, Hoboken, NJ.

Robbins, H. & Monro, S. (1951), 'A stochastic approximation method', *The Annals of Mathematical Statistics* **22**(3), 400–407.

Rockafellar, R. T. & Wets, R. J.-B. (1991), 'Scenarios and policy aggregation in optimization under uncertainty', *Mathematics of Operations Research* **16**(1), 119–147.

Rogers, D., Plante, R., Wong, R. & Evans, J. (1991), 'Aggregation and Disaggregation Techniques and Methodology in Optimization', *Operations Research* **39**, 553–582.

Ross, S. M. (1983), 'Introduction to Stochastic Dynamic Programming', *Academic Press, New York*.

Ruszczyński, A. (1980), 'Feasible Direction Methods for Stochastic Programming Problems', *Math. Programming* **19**, 220–229.

Ruszczyński, A. (1987), 'A Linearization Method for Nonsmooth Stochastic Programming Problems', *Mathematics of Operations Research* **12**, 32–49.

Ruszczyński, A. (1993), 'Parallel Decomposition of Multistage Stochastic Programming Problems', *Mathematical Programming* **58**, 201–228.

Ruszczyński, A. (2003), *Decomposition Methods*, Elsevier, Amsterdam.

Ruszczyński, A. & Shapiro, A. (2003), *Handbooks in Operations Research and Management Science: Stochastic Programming*, Vol. 10, Elsevier, Amsterdam.

Ruszczyński, A. & Syski, W. (1986), 'A method of aggregate stochastic subgradients with on-line stepsize rules for convex stochastic programming problems', *Mathematical Programming Study* **28**, 113–131.

Ryzhov, I. O. & Powell, W. B. (2009), A Monte Carlo Knowledge Gradient Method for Learning Abatement Potential of Emissions Reduction Technologies, *in* M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin & R. G. Ingalls, eds, 'Proceedings of the 2009 Winter Simulation Conference', pp. 1492–1502.

Ryzhov, I. O. & Powell, W. B. (2010), Approximate Dynamic Programming With Correlated Bayesian Beliefs, *in* 'Forty-Eighth Annual Allerton Conference on Communication, Control, and Computing'.

Ryzhov, I. O., Frazier, P. I. & Powell, W. B. (2009), 'Stepsize selection for approximate value iteration and a new optimal stepsize rule', **2009**, 1–31.

Schweitzer, P. & Seidmann, A. (1985), 'Generalized polynomial approximations in Markovian decision processes', *Journal of Mathematical Analysis and Applications* **110**(6), 568–582.

Secomandi, N. (2008), 'An Analysis of the Control-Algorithm Re-solving Issue in Inventory and Revenue Management', *Manufacturing & Service Operations Management* **10**(3), 468–483.

Sen, S. & Higle, J. L. (1999), 'An Introductory Tutorial on Stochastic Linear Programming Models', *Interfaces* (April), 33–61.

Shapiro, A. (2003), *Stochastic Programming*, Vol. 10, John Wiley & Sons, Chichester, U.K.

Singh, S., Jaakkola, T., Littman, M. & Szepesvari, C. (2000), 'Convergence results for single-step on-policy reinforcement-learning algorithms', *Machine Learning* **38**(3), 287—–308.

Singh, S. P., Jaakkola, T. & Jordan, M. I. (1995), 'Reinforcement Learning with Soft State Aggregation', *In Advances in Neural Information Processing Systems, Vol. 7, MIT Press* **pp**, 361–368.

Smola, A. J. & Schölkopf, B. (2004), 'A tutorial on support vector regression', *Statistics and Computing* **14**(3), 199–222.

Soderstrom, T., Ljung, L. & Gustavsson, I. (1978), 'A Theoretical Analysis of Recursive Identification Methods', *Automatica* **78**(3), 231–244.

Stokey, N. L. & R. E. Lucas, J. (1989), *Recursive Methods in Dynamic Economics*, Harvard University Press, Cambridge, MA.

Sutton, R. S. & Barto, A. G. (1998), *Reinforcement Learning*, MIT Press, Cambridge, MA.

Sutton, R. S. & Singh, S. P. (1994), On Step-Size and Bias in Temporal-Difference Learning, *in* C. for System Science, ed., 'Eight Yale Workshop on Adaptive and Learning Systems', Yale University, pp. 91–96.

Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, C. & Wiewiora, E. (2009), 'Fast gradient-descent methods for temporal-difference learning with linear function approximation', *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09* pp. 1–8.

Sutton, R. S., McAllester, D., Singh, S. P. & Mansour, Y. (2000), 'Policy gradient methods for reinforcement learning with function approximation', *Advances in neural information processing systems* **12**(22), 1057–1063.

Sutton, R. S., Singh, S. & Mcallester, D. (1983), 'Comparing Policy-Gradient Algorithms', *IEEE Transactions on Man, Systems and Cybernetics*.

Sutton, R. S., Szepesv, C. & Maei, H. R. (2008), A Convergent O (n) Algorithm for Off-policy Temporal-difference Learning with Linear Function Approximation, *in* 'Proceedings of the Neuro Information Processing Society', Vancouver, pp. 1–8.

Szepesvari, C. (2010), 'Algorithms for Reinforcement Learning', *Synthesis Lectures on Artificial Intelligence and Machine Learning* **4**(1), 1–103.

Taylor, H. (1967), 'Evaluating a call option and optimal timing strategy in the stock market', *Management Science* **12**, 111–120.

Thrun, S. B. (1992), 'The role of exploration in learning control', *In White, D. A., & Sofge, D. A*.

Topaloglu, H. & Powell, W. B. (2003), 'An Algorithm for Approximating Piecewise Linear Concave Functions from Sample Gradients', *Operations Research Letters* **31**, 66–76.

Topkins, D. M. (1978), 'Minimizing a submodular function on a lattice', *Operations Research* **26**, 305–321.

Trigg, D. W. (1964), 'Monitoring a forecasting system', *Operations Research Quarterly* **15**, 271–274.

Tsitsiklis, J. N. (1994), 'Asynchronous Stochastic Approximation and Q-Learning', *Machine Learning* **16**, 185–202.

Tsitsiklis, J. N. & Roy, B. V. (1996), 'Feature-Based Methods for Large Scale Dynamic Programming', *Machine Learning, Vol.* **22**, 59–94.

Tsitsiklis, J. N. & Van Roy, B. (1997), 'An analysis of temporal-difference learning with function approximation', *IEEE Transactions on Automatic Control* **42**(5), 674–690.

Tsitsiklis, J. N. & Van Roy, B. (2001), 'Regression Methods for Pricing Complex American-Style Options', *IEEE Transactions on Neural Networks* **12**, 694–703.

Tsitsiklis, J. N., Roy, V. & B (1997), 'An analysis of temporal-difference learning with function approximation', *IEEE Transactions on Automatic Control* **42**, 674–6905862.

Van Roy, B. (2001), Neuro-Dynamic Programming: Overview and Recent Trends, *in* E. Feinberg & A. Shwartz, eds, 'Handbook of {Markov} Decision Processes: Methods and Applications', Kluwer, Boston, pp. 431–460.

Van Roy, B. & Choi, D. P. (2006), 'A Generalized Kalman Filter for Fixed Point Approximation and Efficient Temporal-Difference Learning', *Discrete Event Dynamic Systems* **16**, 207–239.

Van Roy, B. & de Farias, D. P. (2000), 'On the existence of fixed points for approximate value iteration and temporal-difference learning', *Journal of Optimization Theory and Applications* **105**(3), 589–608.

Venayagamoorthy, G. K., Harley, R. G. & Wunsch, D. G. (n.d.), 'Comparison of heuristic dynamic programming and dual heuristic programming adaptive critics for neurocontrol of a turbogenerator', *IEEE Trans. Neural Networks* **vol**, 13pp764–773.

Wallace, S. W. (1986), 'Decomposing the Requirement Space of a Transportation Problem', *Math. Prog. Study* **28**, 29–47.

Wang, Y., Wang, C., Powell, W. B. & Edu, P. P. (2016), The Knowledge Gradient for Sequential Decision Making with Stochastic Binary Feedbacks, *in* 'ICML-2016', Vol. 48, New York.

Wasan, M. T. (1969), *Stochastic approximation*, Cambridge University Press, Cambridge.

Werbos, P. J. (1992), Approximate Dynamic Programming for Real-Time Control and Neural Modelling, *in* D. J. White & D. A. Sofge, eds, 'Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches'.

White, C. C. (1991), 'A survey of solution techniques for the partially observable Markov decision process', *Annals of operations research* **32**, 215–230.

White, D. J. (1969), *Dynamic Programming*, Holden-Day, San Francisco.

Whitt, W. (1978), 'Approximations of Dynamic Programs I', *Mathematics of Operations Research* **vol**, 231–243.

Whittle, P. (1982), 'Optimization Over Time: Dynamic Programming and Stochastic Control Volumes I and II', *J. Wiley, New York. z̧niakowski, H* **24**, 185–194.

Williams, R. J. (1992), 'Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning', *Machine Learning* **8**, 229–256.

Williams, R. J. & Baird, L. C. (1990), A Mathematical Analysis of Actor-Critic Architectures for Learning Optimal Controls Through Incremental Dynamic Programming, *in* 'Sixth Yale Workshop on Adaotive and Learning Systems', New Haven, pp. 96–101.

Yang, Y. (2001), 'Adaptive Regression by Mixing', *Journal of the American Statistical Association*.

Yao, Y. (2006), Some results on the Gittins index for a normal reward process, *in* H. Ho, C. Ing & T. Lai, eds, 'Time Series and Related Topics: In Memory of Ching-Zong Wei', Institute of Mathematical Statistics, Beachwood, OH, USA, pp. 284–294.

Young, P. (1984), *Recursive Estimation and Time-Series Analysis*, Springer-Verlag, Berlin, Heidelberg.