Applied Computational Economics

Mario J. Miranda
The Ohio State University

and

Paul L. Fackler
North Carolina State University

Contents

Preface

Many interesting economic models cannot be solved analytically using the standard mathematical techniques of Algebra and Calculus. This is often true of applied economic models that attempt to capture the complexities inherent in real-world individual and institutional economic behavior. For example, to be useful in applied economic analysis, the conventional Marshallian partial static equilibrium model of supply and demand must often be generalized to allow for multiple goods, interegional trade, intertemporal storage, and government interventions such as tariffs, taxes, and trade quotas. In such models, the structural economic constraints are of central interest to the economist, making it undesirable, if not impossible, to "assume an internal solution" to render the model analytically tractable.

Another class of interesting models that typically cannot be solved analytically are stochastic dynamic models of rational, forward-looking economic behavior. Dynamic economic models typically give rise to functional equations in which the unknown is not simply a vector in Euclidean space, but rather an entire function defined on a continuum of points. For example, the Bellman and Euler equations that describe dynamic optima are functional equations, as often are the conditions that characterize rational expectations and arbitrage pricing market equilibria. Except in a very limited number of special cases, these functional equations lack a known closed-form solution, even though the solution can be shown theoretically to exist and to be unique.

Models that lack closed-form analytical solution are not unique to economics. Analytically insoluble models are common in biological, physical, and engineering sciences. Since the introduction of the digital computer, scientists in these fields have turned increasingly to numerical computer methods to solve their models. In many cases where analytical approaches fail, numerical methods are often used to successfully compute highly accurate ap-

proximate solutions. In recent years, the scope of numerical applications in the biological, physical, and engineering sciences has grown dramatically. In most of these disciplines, computational model building and analysis is now recognized as a legitimate subdiscipline of specialization. Numerical analysis courses have also become standard in many graduate and undergraduate curriculums in these fields.

Economists, however, have not embraced numerical methods as eagerly as other scientists. Many economists have shunned numerical methods out of a belief that numerical solutions are less elegant or less general than those obtained from algebraic models. The former belief is a subjective, aesthetic judgment that is outside of scientific discourse and beyond the scope of this book. The generality of the results obtained from numerical economic models, however, is another matter. Of course, given an economic model, it is always preferable to derive an explicit algebraic solution—provided such a solution exists. However, when essential features of an economic system being studied cannot be captured neatly in an algebraically soluble model, a choice must be made. Either essential features of the system must be ignored in order to obtain an algebraically tractable model, or numerical techniques must be applied. Too often Economists chose algebraic tractability over Economic realism.

Numerical economic models are often unfairly criticized by economists on the grounds that they rest on specific assumptions regarding functional forms and parameter values. Such criticism, however, is unwarranted when strong empirical support exists for the specific functional form and parameter values used to specify a model. Moreover, even when there is some uncertainty about functional forms and parameters, the model may be solved under a variety of assumptions in order to assess the robustness of its implications. Although some doubt will persist as to the implications of a model outside the range of functional forms and parameter values examined, this uncertainty must be weighed against the lack of relevance of an alternative model that is algebraically soluble, but which ignores essential features of the economic system of interest. We believe that it is better to derive economic insights from a realistic numerical model of an economic system than to derive irrelevant results, however general, from an unrealistic, but tractable algebraic model.

Despite the resistance placed by the economics profession as a whole, an increasing number of economists are becoming aware of the potential benefits of numerical economic model building and analysis. This is evidenced

by the recent introduction of journals and an economic society devoted to the sub-discipline of computational economics. The growing popularity of computational economics, however, has been impeded by the absence of adequate textbooks and computer software. The methods of numerical analysis and much of the available computer software have been largely developed for non-economic disciplines, most notably the physical, mathematical, and computer sciences. The scholarly literature can also pose substantial barriers for economists, both because of its mathematical prerequisites and because its examples are unfamiliar to economists. Many available software packages, moreover, are designed to solve problems that are specific to the physical sciences.

This book attempts to address, in a number of ways, the difficulties typically encountered by economists attempting to learn and apply numerical methods. First, this book emphasizes practical numerical methods, not mathematical proofs, and focuses on techniques that will be directly useful to economic analysts, not those that would be useful exclusively to physical scientists. Second, the examples used in the book are drawn from a wide range of sub-specialties of economics and finance, both in macro- and microeconomics, with particular emphasis on problems in agricultural, financial, environmental, and macro- economics. And third, we include with the textbook a library of computer utilities and demonstration programs to provide interested researchers with a starting point for their own computer models.

We make no attempt to be encyclopedic in our coverage of numerical methods or potential economic applications. We have instead chosen to develop only a relatively small number of techniques that can be applied easily to a wide variety of economic problems. In some instances, we have deviated from the standard treatments of numerical methods in existing textbooks in order to present a simple consistent framework that may be readily learned and applied by economists. In many cases we have elected not to cover certain numerical techniques when we regard them to be of limited benefit to economists, relative to their complexity. Throughout the book, we try to explain our choices clearly and to give references to more advanced numerical textbooks where appropriate.

The book is divided into two major sections. In the first seven chapters, we develop basic numerical methods, including root finding, complementarity, finite-dimensional optimization, numerical integration, and function approximation methods. In these chapters, we develop appreciation for basic numerical techniques by illustrating their application to partial equilibrium

and optimization models familiar to most economists. The last five chapters of the book are devoted to methods for solving and estimating dynamic stochastic models in economic and finance, including dynamic programming, rational expectations, and arbitrage pricing models in discrete and continuous time.

The book is aimed at both graduate students, advanced undergraduate students, and practicing economists. We have attempted to write a book that can be used both as a classroom text and for self-study. We have also attempted to make the various sections reasonably self-contained. For example, the sections on discrete time continuous state models are largely independent from those on discrete time discrete state models. Although this results in some duplication of material, we felt that this would increase the usefulness of the text by allowing readers to skip sections.

Although we have attempted to keep the mathematical prerequisites for this book to a minimum, some mathematical training and insight is necessary to work with computational economic models and numerical techniques. We assume that the reader is familiar with ideas and methods of linear algebra and calculus. Appendix A provides an overview of the basic mathematics used throughout the text. Furthermore, in an attempt to make the book modular in organization, some of the mathematics used in studying specific classes of dynamic models is developed in the text as needed. Examples include the basic theory of Markov processes, dynamic programming, and, for continuous time models, Ito stochastic calculus.

One barrier to the use of numerical methods by economists is lack of access to functioning computer code. This presents an apparent dilemma to us as textbook authors, given the variety of computer languages available. On the one hand, it is useful to have working examples of code in the book and to make the code available to readers for immediate use. On the other hand, using a specific language in the text could obscure the essence of the numerical routines for those unfamiliar with the chosen language. We believe, however, that the latter concern can be substantially mitigated by conforming to the syntax of a vector processing language. Vector processing languages are designed to facilitate numerical analysis and their syntax is often simple enough that the language is transparent and easily learned and implemented. Due to its facility of use and its wide availability on university campus computing systems, we have chosen to illustrate algorithms in the book using Matlab and have provided an extensive library of Matlab utilities and demonstration programs to assist interested readers develop their own

computational economic applications. In the future, we plan to make available these programs available in other popular languages, including Gauss and Fortran.

Our ultimate goal in writing this book is to motivate a broad range of economists to use numerical methods in their work by demonstrating the essential principles underlying computational economic models across subdisciplines. It is our hope that this book will help broaden the scope of economic analysis by helping economists to solve economic and financial models that heretofore they were unable to solve within the confines of traditional mathematical economic analysis.

Chapter 1

Introduction

1.1 Some Apparently Simple Questions

Consider the constant elasticity demand function

$$q = p^{-0.2}.$$

This is a function because for each price p there is an unique quantity demanded q. Given a hand-held calculator, any economist could easily compute the quantity demanded at any given price.

An economist would also have little difficulty computing the price that clears the market of a given quantity. Flipping the demand expression about the equality sign and raising each side to the power of -5, the economist would derive a closed-form expression for the inverse demand function

$$p = q^{-5}.$$

Again, using a calculator any economist could easily compute the price that will exactly clear the market of any given quantity.

Suppose now that the economist is presented with a slightly different demand function

$$q = 0.5 \cdot p^{-0.2} + 0.5 \cdot p^{-0.5},$$

one that is the sum a domestic demand term and an export demand term. Using standard calculus, the economist could easily verify that the demand function is continuous, differentiable, and strictly decreasing. The economist once again could easily compute the quantity demanded at any price using

a calculator and could easily and accurately draw a graph of the demand function.

However, suppose that the economist is asked to find the price that clears the market of, say, a quantity of 2 units. The question is well-posed. A casual inspection of the graph of the demand function suggests that its inverse is well-defined, continuous, and strictly decreasing. A formal argument based on the Intermediate Value and Implicit Function Theorems would prove that this is so. An unique market clearing price clearly exists.

But what is the inverse demand function? And what price clears the market? After considerable effort, even the best trained economist will not find an answer using Algebra and Calculus. No apparent closed-form expression for the inverse demand function exists. The economist cannot answer the apparently simple question of what the market clearing price will be.

Consider now a simple model of an agricultural commodity market. In this market, acreage supply decisions are made before the per-acre yield and harvest price are known. Planting decisions are based on the price expected at harvest:

$$a = 0.5 + 0.5 \cdot Ep.$$

After the acreage is planted, a random yield $\tilde{y}$ is realized, giving rise to a supply

$$q = a \cdot \tilde{y}.$$

The supply is entirely sold at a market clearing price

$$p = 3 - 2q.$$

Yield is exogenous and distributed normally with a mean of 1 and a variance of 0.1.

Most economists would have little difficulty deriving the rational expectations equilibrium of this market model. Substituting the first expression into the second, and then the second into the third, the economist would write

$$p = 3 - 2(0.5 + 0.5 \cdot Ep) \cdot \tilde{y}.$$

Taking expectations on both sides

$$Ep = 3 - 2(0.5 + 0.5 \cdot Ep),$$

she would solve for the equilibrium expected price $Ep = 1$. She would conclude that the equilibrium acreage is $a = 1$ and the equilibrium price distribution has a standard deviation of 0.4.

Suppose now that the economist is asked to assess the implications of a proposed government price support program. Under this program, the government guarantees each producer a minimum price, say 1. If the market price falls below this level, the government simply pays the producer the difference per unit produced. The producer thus receives an effective price of $\max(p, 1)$ where p is the prevailing market price. The government program transforms the acreage supply relation to

$$a = 0.5 + 0.5 \cdot E \max(p, 1).$$

Before proceeding with a formal mathematical analysis, the economist exercises a little economic intuition. The government support, she reasons, will stimulate acreage supply, raising acreage planted. This will shift the equilibrium price distribution to the left, reducing the expected market price below 1. Price would still occasionally rise above 1, however, implying that the expected effective producer price will exceed 1. The difference between the expected effective producer price and the expected market price represents a positive expected government subsidy.

The economist now attempts to formally solve for the rational expectations equilibrium of the revised market model. She performs the same substitutions as before and writes

$$p = 3 - 2(0.5 + 0.5 \cdot E \max(p, 1)) \cdot \tilde{y}.$$

As before, she takes expectations on both sides

$$Ep = 3 - 2(0.5 + 0.5 \cdot E \max(p, 1)).$$

In order to solve the expression for the expected price, the economist uses a fairly common and apparently innocuous trick: she interchanges the max and E operators, replacing $E \max(p, 1)$ with $\max(Ep, 1)$. The resulting expression is easily solved for $Ep = 1$. This solution, however, asserts the expected market price and acreage planted remain unchanged by the introduction of the government price support policy. This is inconsistent with the economist's intuition.

The economist quickly realizes her error. The expectation operator cannot be interchanged with the maximization operator because the latter is

a nonlinear function. But if this operation is not valid, then what mathematical operations would allow the economist to solve for the equilibrium expected price and acreage?

Again, after considerable effort, our economist is unable to find an answer using Algebra and Calculus. No apparent closed-form solution exists for the model. The economist cannot answer the apparently simple question of how the equilibrium acreage and expected market price will change with the introduction of the government price support program.

1.2 An Alternative Analytic Framework

The two problems discussed in the preceding section illustrate how even simple economic models cannot always be solved using standard mathematical techniques. These problems, however, can easily be solved to a high degree of accuracy using numerical methods.

Consider the inverse demand problem. An economist who knows some elementary numerical methods and who can write basic Matlab code would have little difficulty solving the problem. The economist would simply write the following elementary Matlab program:

```
p = 0.25;
for i=1:100
  deltap = (.5*p^-.2+.5*p^-.5-2)/(.1*p^-1.2 + .25*p^-1.5);
  p = p + deltap;
  if abs(deltap) < 1.e-8, break, end
end
disp(p);
```

He would then execute the program on a computer and, in an instant, compute the solution: the market clearing price is 0.154. The economist has used Newton's rootfinding method.

Consider now the rational expectations commodity market model with government intervention. The source of difficulty in solving this problem is the need to evaluate the truncated expectation of a continuous distribution. An economist who knows some numerical analysis and who knows how to write basic Matlab code, however, would have little difficulty computing the rational expectation equilibrium of this model. The economist would replace the original normal yield distribution with a discrete distribution that has identical lower moments, say one that assumes values $y_1, y_2, \ldots, y_n$

with probabilities $w_1, w_2, \ldots, w_n$. After constructing the discrete distribution approximant, which would require only a single call to a library routine, call it `qnorm`, the economist would code and execute the following elementary Matlab program:[1]

```
[y,w] = qnwnorm(10,1,0.1);
a = 1;
for it=1:100
  aold = a;
  p = 3 - 2*a*y;
  f = w'*max(p,1);
  a = 0.5 + 0.5*f;
  if abs(a-aold)<1.e-8, break, end
end
disp(a);disp(f);disp(w'*p)
```

In an instant, the program would compute and display the rational expectations equilibrium acreage, 1.10, the expected market price, 0.81, and the expected effective producer price, 1.19. The economist has combined Gaussian quadrature techniques and fixed-point function iteration methods to solve the problem.

[1]The `qnorm`, is discussed in Chapter 4.

Chapter 2

Linear Equations

The *linear equation* is the most elementary problem that arises in computational economic analysis. In a linear equation, an $n \times n$ matrix A and an n-vector b are given, and one must compute the n-vector x that satisfies

$$Ax = b.$$

Linear equations arise, directly or indirectly, in most computational economic applications. For example, a linear equation may be solved when computing the steady-state distribution of a discrete-state stochastic economic process or when computing the equilibrium prices and quantities of a multicommodity market model with linear demand and supply functions. Linear equations also arise as elementary tasks in solution procedures designed to solve more complicated nonlinear economic models. For example, a nonlinear partial equilibrium market model may be solved using Newton's method, which involves solving a sequence of linear equations. And the Euler functional equation of a rational expectations model may be solved using a collocation method, which yields a nonlinear equation that in turn is solved as a sequence of linear equations.

Various practical issues arise when solving a linear equation numerically. Digital computers are capable of representing arbitrary real numbers with only limited precision. Numerical arithmetic operations, such as computer addition and multiplication, produce rounding errors that may, or may not, be negligible. Unless the rounding errors are controlled in some way, the errors can accumulate, rendering a computed solution that may be far from correct. Speed and storage requirements are also important considerations in the design of a linear equation solution algorithm. In some applications, such

as the stochastic simulation of a rational expectations model, linear equations may have to be solved millions of times. And in other applications, such as computing option prices using finite difference methods, linear equations with a very large number of variables and equations may be encountered.

Over the years, numerical analysts have studied linear equations extensively and have developed algorithms for solving them quickly, accurately, and with a minimum of computer storage. In most applied work, one can typically rely on Gaussian elimination, which may be implemented in various different forms depending on the structure of the linear equation. Iterative methods offer an alternative to Gaussian elimination and are especially efficient if the A matrix is large and consists mostly of zero entries.

2.1 L-U Factorization

Some linear equations $Ax = b$ are relatively easy to solve. For example, if A is a lower triangular matrix,

$$A = \begin{bmatrix} a_{11} & 0 & 0 & \dots & 0 \\ a_{21} & a_{22} & 0 & \dots & 0 \\ a_{31} & a_{32} & a_{33} & \dots & 0 \\ & & & & \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix},$$

then the elements of x can be computed recursively using *forward-substitution*:

$$\begin{aligned} x_1 &= b_1/a_{11} \\ x_2 &= (b_2 - a_{21}x_1)/a_{22} \\ x_3 &= (b_3 - a_{31}x_1 - a_{32}x_2)/a_{33} \\ &\vdots \\ x_n &= (b_n - a_{n1}x_1 - a_{n2}x_2 - \ldots - a_{nn-1}x_{n-1})/a_{nn}. \end{aligned}$$

This may be written more compactly using summation notation as

$$x_i = \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j \right) / a_{ii} \qquad \forall i.$$

In the vector processing language Matlab, this may be implemented as follows:

```
for i=1:length(b)
  x(i)=(b(i)-A(i,1:i-1)*x(1:i-1))/A(i,i);
end
```

If A is an upper triangular matrix, then the elements of x can be computed recursively using *backward-substitution.*

Most linear equations encountered in practice, however, do not have a triangular A matrix. In such cases, the linear equation is often best solved using the *L-U factorization algorithm.* The L-U algorithm is designed to decompose the A matrix into the product of lower and upper triangular matrices, allowing the linear equation to be solved using a combination of backward and forward substitution.

The L-U algorithm involves two phases. In the *factorization* phase, Gaussian elimination is used to factor the matrix A into the product

$$A = LU$$

of a row-permuted lower triangular matrix L and an upper triangular matrix U. A row-permuted lower triangular matrix is simply a lower triangular matrix that has had its rows rearranged. Any nonsingular square matrix can be decomposed in this way.

In the *solution* phase of the L-U algorithm, the factored linear equation

$$Ax = (LU)x = L(Ux) = b$$

is solved by first solving

$$Ly = b$$

for y using forward substitution, accounting for row permutations, and then solving

$$Ux = y$$

for x using backward substitution.

Consider, for example, the linear equation $Ax = b$ where

$$A = \begin{bmatrix} -3 & 2 & 3 \\ -3 & 2 & 1 \\ 3 & 0 & 0 \end{bmatrix} \text{ and } b = \begin{bmatrix} 10 \\ 8 \\ -3 \end{bmatrix}.$$

The matrix A can be decomposed into the product $A = LU$ where

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ -1 & 1 & 0 \end{bmatrix} \text{ and } U = \begin{bmatrix} -3 & 2 & 3 \\ 0 & 2 & 3 \\ 0 & 0 & -2 \end{bmatrix}.$$

The matrix L is row-permuted lower triangular because upon interchanging the second and third rows, a lower diagonal matrix results. The matrix U is upper triangular. Solving $L * y = b$ for y using forward substitution involves first solving for y_1, then for y_3, and finally for y_2. Given the solution $y = (10, 7, -2)'$, the linear equation $Ux = y$ can the be solved using backward substitution, yielding the solution of the original linear equation, $x = (-1, 2, 1)$.

The L-U factorization algorithm is faster than other linear equation solution methods that are typically presented in elementary linear algebra courses. For large n, it takes approximately $n^3/3 + n^2$ long operations (multiplications and divisions) to solve an $n \times n$ linear equation using L-U factorization. Explicitly computing the inverse of A and then computing $A^{-1}b$ requires approximately $n^3 + n^2$ long operations. Solving the linear equation using Cramer's rule requires approximately $(n+1)!$ long operations. To solve a 10×10 linear equation, for example, L-U factorization requires exactly 430 long operations, whereas matrix inversion and multiplication requires exactly 1100 long operations and Cramer's rule requires nearly 40 million long operations.

Linear equations arise so frequently in numerical analysis that most numerical subroutine packages and software programs include either a basic subroutine or an intrinsic function for solving a linear equation using L-U factorization. In Matlab, the solution to the linear equation $Ax = b$ is returned by the statement $x = A \setminus b$. The '\', or "backslash", operator is designed to solve the linear equation using L-U factorization, unless a special structure for A is detected, in which case Matlab may implicitly use another, more efficient method. In particular, if Matlab detects that A is triangular or permuted triangular, it will dispense with L-U factorization and solve the linear equation directly using forward or backward substitution. Matlab also uses special algorithms when the A matrix is positive definite.

Although L-U factorization is the best general method for solving a linear equation, situations can arise in which alternative methods may be preferable. For example, in many computational economic applications, one must solve

a series of linear equations, all having the same A matrix, but different b vectors, $b_1, b_2, \ldots, b_m$. In this situation, it is often computationally more efficient to directly compute and store the inverse of A first and then compute the solutions $x = A^{-1}b_j$ by performing only direct matrix-vector multiplications. Whether explicitly computing the inverse is faster than L-U factorization depends on the size of the linear equation system n and the number of times, m, an equation system is to be solved. Computing $x = A \setminus b_j$ a total of m times involves $\frac{mn^3}{3} + mn^2$ long operations. Computing A^{-1} once and then computing $A^{-1}b_j$ a total of m times requires $n^3 + mn^2$ long operations. Thus explicit computation of the inverse should be faster than L-U factorization whenever the number of equations to be solved m is greater than three or four. The actual breakeven point will vary across numerical analysis packages, depending on the computational idiosyncrasies and overhead costs of the L-U factorization and inverse routines implemented in the package.

2.2 Gaussian Elimination

The L-U factors of a matrix A are computed using *Gaussian elimination*. Gaussian elimination is based on two elementary row operations: subtracting a constant multiple of one row of a linear equation from another row, and interchanging two rows of a linear equation. Either operation may be performed on a linear equation without altering its solution.

The Gaussian elimination algorithm begins with matrices L and U initialized as $L = I$ and $U = A$, where I is the identity matrix. The algorithm then uses elementary row operations to transform U into an upper triangular matrix, while preserving the permuted lower diagonality of L and the factorization $A = LU$:

Consider the matrix

$$A = \begin{bmatrix} 2 & 0 & -1 & 2 \\ 4 & 2 & -1 & 4 \\ 2 & -2 & -2 & 3 \\ -2 & 2 & 7 & -3 \end{bmatrix}.$$

The first stage of Gaussian elimination is designed to nullify the subdiagonal entries of the first column of the U matrix. The U matrix is updated by subtracting 2 times first row from the second, subtracting 1 times the first row from the third, and subtracting -1 times the first row from the fourth.

The L matrix, which initially equals the identity, is updated by storing the multipliers 2, 1, and -1 as the subdiagonal entries of its first column. These operations yield updated L and U matrices:

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix} \qquad U = \begin{bmatrix} 2 & 0 & -1 & 2 \\ 0 & 2 & 1 & 0 \\ 0 & -2 & -1 & 1 \\ 0 & 2 & 6 & -1 \end{bmatrix}.$$

After the first stage of Gaussian elimination, $A = LU$ and L is lower triangular, but U is not yet upper triangular.

The second stage Gaussian elimination is designed to nullify the subdiagonal entries of the second column of the U matrix. The U matrix is updated by subtracting -1 times second row from the third and subtracting 1 times the second row from the fourth. The L matrix is updated by storing the multipliers -1 and 1 as the subdiagonal elements of its second column. These operations yield updated L and U matrices:

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 1 & -1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{bmatrix} \qquad U = \begin{bmatrix} 2 & 0 & -1 & 2 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 5 & -1 \end{bmatrix}.$$

After the second stage of Gaussian elimination, $A = LU$ and L is lower triangular, but U still is not upper triangular.

In the third stage of Gaussian elimination, one encounters an apparent problem. The third diagonal element of the matrix U is zero, making it impossible to nullify the subdiagonal entry as before. This difficulty is easily remedied, however, by interchanging the third and fourth rows of U. The L matrix is updated by interchanging the previously computed multipliers residing in the third and fourth rows. These operations yield updated L and U matrices:

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 1 & -1 & 0 & 1 \\ -1 & 1 & 1 & 0 \end{bmatrix} \qquad U = \begin{bmatrix} 2 & 0 & -1 & 2 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 5 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The Gaussian elimination algorithm terminates with a permuted lower triangular matrix L and an upper triangular matrix U whose product is the

matrix A. In theory, Gaussian elimination will compute the L-U factors of any matrix A, provided A is invertible. If A is not invertible, Gaussian elimination will detect this by encountering a zero diagonal element in the U matrix that cannot be replaced with a nonzero element below it.

2.3 Rounding Error

In practice, Gaussian elimination performed on a computer can sometimes render inaccurate solutions due to rounding errors. The effects of rounding errors, however, can often be controlled by *pivoting*.

Consider the linear equation

$$\begin{bmatrix} -L^{-1} & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}.$$

where L is a large positive number.

To solve this equation via Gaussian elimination, a single row operation is required: subtracting $-L$ times the first row from the second row. In principle, this operation yields the L-U factorization

$$\begin{bmatrix} -L^{-1} & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -L & 1 \end{bmatrix} \begin{bmatrix} -L^{-1} & 1 \\ 0 & L+1 \end{bmatrix}.$$

In theory, applying forward and backward substitution yields the solution $x_1 = L/(L+1)$ and $x_2 = (L+2)/(L+1)$, which are both very nearly one.

In practice, however, Gaussian elimination may yield a very different result. In performing Gaussian elimination, one encounters an operation that cannot be carried out precisely on a computer, and which should be avoided in computational work: adding or subtracting values of vastly different magnitudes. On a computer, it is not meaningful to add or subtract two values whose magnitude differ by more than the number of significant digits that the computer can represent. If one attempts such an operation, the smaller value is effectively treated as zero. For example, the sum of 0.1 and 0.0001 may be 0.1001, but on a hypothetical machine with three digit precision the result of the sum is rounded to 0.1 before it is stored.

In the linear equation above, adding 1 or 2 to a sufficiently large L on a computer simply returns the value L. Thus, in the first step of the backward substitution, x_2 is computed, not as $(L+2)/(L+1)$, but rather as L/L, which is exactly one. Then, in the second step of backward substitution,

$x_1 = -L(1 - x_2)$ is computed to be zero. Rounding error thus produces computed solution for x_1 that has a relative error of nearly 100 percent.

Fortunately, there is a partial remedy for the effects of rounding error in Gaussian elimination. Rounding error arises in the example above because the diagonal element $-L^{-1}$ is very small. Interchanging the two rows at the outset of Gaussian elimination does not alter the theoretical solution to the linear equation, but allows one to perform Gaussian elimination with a diagonal element of larger magnitude.

Consider the equivalent linear equation system after the rows have been interchanged:

$$\begin{bmatrix} 1 & 1 \\ -L^{-1} & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}.$$

After interchanging the rows, the new A matrix may be factored as

$$\begin{bmatrix} 1 & 1 \\ -L^{-1} & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -L^{-1} & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & L^{-1} + 1 \end{bmatrix}.$$

Backward and forward substitution yield the theoretical results $x_1 = 1 - L^{-1}$ and $x_2 = L^{-1} + 1 + L^{-1}(1 - L^{-1})$. In evaluating these expressions on the computer, one again encounters rounding error. Here, x_2 is numerically computed to be exactly one as before. However, x_1 is also computed to be exactly one. The computed solution, though not exactly correct, is correct to the precision available on the computer, and is certainly more accurate than the one obtained without interchanging the rows.

Interchanging rows during Gaussian elimination in order to make the magnitude of diagonal element as large as possible is called pivoting. Pivoting substantially enhances the reliability and the accuracy of a Gaussian elimination routine. For this reason, all good Gaussian elimination routines designed to perform L-U factorization, including the ones implemented in Matlab, employ some form of pivoting.

2.4 Ill Conditioning

Pivoting cannot cure all the problems caused by rounding error. Some linear equations are inherently difficult to solve accurately on a computer, despite pivoting. This occurs when the A matrix is structured in such a way that a small perturbation δb in the data vector b induces a large change δx in the

solution vector x. In such cases the linear equation or, more generally, the A matrix are said to be *ill-conditioned*.

One measure of ill-conditioning in a linear equation $Ax = b$ is the "elasticity" of the solution vector x with respect to the data vector b

$$\epsilon = \sup_{||\delta b||>0} \frac{||\delta x||/||x||}{||\delta b||/||b||}.$$

The elasticity gives the maximum percentage change in the size of the solution vector x induced by a one percent change the size of the data vector b. If the elasticity is large, then small errors in the computer representation of the data vector b can produce large errors in the computed solution vector x. Equivalently, the computed solution x will have far fewer significant digits than the data vector b.

The elasticity of the solution is expensive to compute and thus is virtually never computed in practice. In practice, the elasticity is estimated using the condition number of the matrix A, which for invertible A is defined by

$$\kappa \equiv ||A|| \cdot ||A^{-1}||.$$

The condition number of A is the least upper bound of the elasticity. The bound is tight in that for some data vector b, the condition number equals the elasticity. The condition number is always greater than or equal to one. Numerical analysts often use the rough rule of thumb that for each power of 10 in the condition number, one significant digit is lost in the computed solution vector x. Thus, if A has a condition number of 1000, the computed solution vector x will have about three fewer significant digits than the data vector b.

Consider the linear equation $Ax = b$ where $A_{ij} = i^{n-j}$ and $b_i = (i^n - 1)/(i - 1)$. In theory, the solution x to this linear equation is a vector containing all ones for any n. In practice, however, if one solves the linear equation numerically using Matlab's '\' operator one can get quite different results. Below is a table that gives the supremum norm approximation error in the computed value of x and the condition number of the A matrix for different n:

n	Approximation Error	Condition Number
5	2.5e-013	2.6e+004
10	5.2e-007	2.1e+012
15	1.1e+002	2.6e+021
20	9.6e+010	1.8e+031
25	8.2e+019	4.2e+040

In this example, the computed answers are accurate to seven decimal up to $n = 10$. The accuracy, however, deteriorates rapidly after that. In this example, the matrix A is a member of the a class of notoriously ill-conditioned matrices called the Vandermonde matrices, which can arise in applied numerical work if one is not careful.

Ill-conditioning ultimately can be ascribed to the limited precision of computer arithmetic. The effects of ill-conditioning can often be mitigated by performing computer arithmetic using the highest precision available on the computer. The best way to handle ill-conditioning, however, is to avoid it altogether. This is often possible when the linear equation problem is as an elementary task in a more complicated solution procedure, such as solving a nonlinear equation or approximating a function with a polynomial. In such cases one can sometimes reformulate the problem or alter the solution strategy to avoid the ill-conditioned linear equation. We will see several examples of this avoidance strategy later in the book.

2.5 Special Linear Equations

Gaussian elimination can be accelerated for A matrices possessing certain special structures. Two classes of A matrices that arise frequently in computational economic analysis and for which such an acceleration is possible are symmetric positive definite matrices and sparse matrices.

Linear equations $Ax = b$ in which A is a symmetric positive definite arise frequently in least-squares curve-fitting and optimization applications. A special form of Gaussian elimination, the Cholesky factorization algorithm, may be applied to such linear equations. Cholesky factorization requires only half as many operations as general Gaussian elimination and has the added advantage that it is less vulnerable to rounding error and does not require pivoting.

The essential idea underlying Cholesky factorization is that any symmetric positive definite matrix A can be uniquely expressed as the product

$$A = U'U$$

of an upper triangular matrix U and its transpose. The matrix U is called the Cholesky factor or square root of A. Given the Cholesky factor of A, the linear equation

$$Ax = U'Ux = U'(Ux) = b$$

may be solved efficiently by using forward substitution to solve

$$U'y = b$$

and then using backward substitution to solve

$$Ux = y.$$

The Matlab '\' operator will automatically employ Cholesky factorization, rather than L-U factorization, to solve the linear equation if it detects that A is symmetric positive definite.

Another situation that often arises in computational practice are linear equations $Ax = b$ in which the A matrix is sparse, that is, it consists largely of zero entries. For example, in solving differential equations, one often encounters tridiagonal matrices, which are zero except on or near the diagonal. When the A matrix is sparse, the conventional Gaussian elimination algorithm consists largely of meaningless, but costly, operations involving either multiplication or addition with zero. The Gaussian elimination algorithm in these instances can often be dramatically increased by avoiding these useless operations.

Matlab has special routines for efficiently storing sparse matrices and operating with them. In particular, the Matlab command `S=sparse(A)` creates a version S of the matrix A stored in a sparse matrix format, in which only the nonzero elements of A and their indices are explicitly stored. Sparse matrix storage requires only a fraction of the space required to store A in standard form if A is sparse. Also, the operator '\' is designed to recognize whether a sparse matrix is involved in the operation and adapts the Gaussian elimination algorithm to exploit this property. In particular, both $x = S \backslash b$ and $x = A \backslash b$ will compute the answer to $Ax = b$. However, the former expression will be executed substantially faster by avoiding meaningless operations with zeros.

2.6 Iterative Methods

Algorithms based on Gaussian elimination are called *exact* or, more properly, *direct methods* because they would generate exact solutions for the linear equation $Ax = b$ after a finite number of operations, if not for rounding error. Such methods are ideal for moderately-sized linear equations, but may be impractical for large ones. Other methods, called *iterative methods* can often be used to solve large linear equations more efficiently if the A matrix is sparse, that is, if A is composed mostly of zero entries. Iterative methods are designed to generate a sequence of increasingly accurate approximations to the solution of a linear equation, but generally do not yield an exact solution after a prescribed number of steps, even in theory.

The most widely-used iterative methods for solving a linear equation $Ax = b$ are developed by choosing an easily invertible matrix Q and writing the linear equation in the equivalent form

$$Qx = b + (Q - A)x$$

or

$$x = Q^{-1}b + (I - Q^{-1}A)x.$$

This form of the linear equation suggests the iteration rule

$$x^{(k+1)} \leftarrow Q^{-1}b + (I - Q^{-1}A)x^{(k)},$$

which, if convergent, must converge to a solution of the linear equation.

Ideally, the so-called *splitting matrix* Q will satisfy two criteria. First, $Q^{-1}b$ and $Q^{-1}A$ should be relatively easy to compute. This is true if Q is either diagonal or triangular. Second, the iterates should converge quickly to the true solution of the linear equation. If

$$||I - Q^{-1}A|| < 1$$

in any matrix norm, then the iteration rule is a contraction mapping and is guaranteed to converge to the solution of the linear equation from any initial value. The smaller the value of the matrix norm $||I - Q^{-1}A||$, the faster the guaranteed rate of convergence of the iterates when measured in the associated vector norm.

The two most popular iterative methods are the Gauss-Jacobi and Gauss-Seidel methods. The Gauss-Jacobi method sets Q equal to the diagonal

matrix formed from the diagonal entries of A. The Gauss-Seidel method sets Q equal to the upper triangular matrix formed from the upper triagonal elements of A. Using the row-sum matrix norm to test the convergence criterion, both methods are guaranteed to converge from any starting value if A is diagonally dominant, that is, if

$$|A_{ii}| > \sum_{\substack{i=1 \\ i \neq j}}^{n} |A_{ij}| \qquad \forall i.$$

Diagonally dominant matrices arise naturally in many computational economic applications, including the solution of differential equations and the approximation of functions using cubic splines, both of which will be discussed in later sections.

The following Matlab script solves the linear equation $Ax = b$ using Gauss-Jacobi iteration:

```
d = diag(A);
for it=1:maxit
  dx = (b-A*x)./d;
  x  = x+dx;
  if norm(dx)<tol, break, end
end
```

Here, the user specifies the data `A` and `b` and an initial guess x for the solution of the linear equation, typically the zero vector or b. Iteration continues until the norm of the change `dx` in the iterate falls below the specified convergence tolerance *tol* or until a specified maximum number of allowable iterations `maxit` are performed.

The following Matlab script solves the same linear equation using Gauss-Seidel iteration:

```
Q = tril(A);
for it=1:maxit
  dx = Q\(b-A*x);
  x  = x+lambda*dx;
  if norm(dx)<tol, break, end
end
```

Here, we have incorporated a so-called *over-relaxation parameter*, λ. Instead of using $x + dx$, we use $x + \lambda dx$ to compute the next iterate. It is often true, though not universally so, that a value of λ between 1 and 2 will accelerate convergence of the Gauss-Seidel algorithm.

The Matlab subroutine library accompanying the textbook includes functions `gjacobi` and `gseidel` that solve linear equations using Gauss-Jacobi and Gauss-Seidel iteration, respectively. The following script solves a linear equation using Gauss-Seidel iteration with default value of 1 for the over-relaxation parameter:

```
A = [3 1 ; 2 5];
b = [7 ; 9];
x = gseidel(A,b)
```

Execution of this script produces the result `x=[2;1]`. When `A=[3 2; 4 1]`, however, the algorithm diverges. The subroutines are extensible in that they allow the user to override the default values of the convergence parameters and, in the case of `gseidel`, the default value of the over-relaxation parameter.

A general rule of thumb is that if A is large and sparse, then the linear equation is a good candidate for iterative methods, provided that sparse matrix storage functions are used to reduce storage requirements and computational effort. Iterative methods, however, have some drawbacks. First, iterative methods, in contrast to direct methods, can fail to converge. Furthermore, it is often difficult or computationally costly to check whether a specific problem falls into a class of problems known to be convergent. It is therefore always a good idea to monitor whether the iterations seem to be diverging and try something else if they are. Second, satisfaction of the termination criteria do not necessarily guarantee a similar level of accuracy in the solution, as measured as the deviation of the approximate solution from the true (but unknown) solution.

Exercises

1. Plot the function $f(x) = 1 - e^{2x}$ on the interval $[-1, 1]$ using a grid of evenly-spaced points 0.01 units apart.

2. Consider the matrices

$$A = \begin{bmatrix} 0 & -1 & 2 \\ -2 & -1 & 4 \\ 2 & 7 & -3 \end{bmatrix}$$

and

$$B = \begin{bmatrix} -7 & 1 & 1 \\ 7 & -3 & -2 \\ 3 & 5 & 0 \end{bmatrix}$$

and the vector

$$y = \begin{bmatrix} 3 \\ -1 \\ 2 \end{bmatrix}.$$

(a) Formulate the *standard* matrix product $C = A * B$ and solve the linear equation $Cx = y$. What are the values of C and x?

(b) Formulate the *element-by-element* matrix product $C = A.*B$ and solve the linear equation $Cx = y$. What are the values of C and x?

3. Using the Matlab standard normal pseudo-random number generator "randn", simulate a hypothetical time series $\{y_t\}$ governed by the structural relationship

$$y_t = 5 + 0.05t + \epsilon_t$$

for years $t = 1960, 1961, \ldots, 2000$, assuming that the ϵ_t are independently and identically distributed with mean 0 and standard deviation 0.2. Using only Matlab elementary matrix operations, regress the simulated observations of y_t on a constant and time, then plot the actual values of y and estimated trend line against time.

4. Consider a stationary 3-state Markov chain with transition probability matrix:

$$P = \begin{bmatrix} 0.2 & 0.4 & 0.4 \\ 0.5 & 0.5 & 0.0 \\ 0.6 & 0.2 & 0.2 \end{bmatrix}.$$

(a) Is the Markov chain irreducible?

(b) If so, find the steady-state distribution.

You may wish to refer to Appendix A.4 for an introduction to Markov chain.

5. Solve $Ax = b$ for

$$A = \begin{bmatrix} 54 & 14 & -11 & 2 \\ 14 & 50 & -4 & 29 \\ -11 & -4 & 55 & 22 \\ 2 & 29 & 22 & 95 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

by

   (a) L-U decomposition

   (b) Gauss-Jacobi iteration

   (c) Gauss-Seidel iteration

   How many Gauss-Jacobi and Gauss-Seidel iterations are required to get answers that agree with the L-U decomposition solution to four significant digits?

6. Use the Matlab function `randn` to generate a random 10 by 10 matrix A and a random 10-vector b. Then use the Matlab function `flop` to count the number of floating point operations needed to solve the linear equation $Ax = b$ 1, 10, and 50 times for each of the following algorithms:

   (a) $x = A \setminus b$

   (b) $x = U \setminus (L \setminus b)$, computing the L-U factors of A only once using the Matlab function `lu`.

   (c) $x = A^{-1}b$, computing A^{-1} only once using the Matlab function `inv`.

7. Consider the rational expectations commodity market model of Chapter 1, except now assume that the yield has a simple two point distribution in which yields of 0.7 and 1.3 are equally probable.

   (a) Compute the expectation and variance of price without government support payments.

   (b) Compute the expectation and variance of the effective producer price assuming a support price of 1.

(c) What is the expected government subsidy per planted acre?

8. Dairy cows can produce milk over 6 lactation cycles. The probability of replacing a cow with a new one after each cycle is given by

Cycle	Prob
1	0.03
2	0.04
3	0.12
4	0.39
5	0.80
6	1.00

(a) What are the proportions of dairy cows in each lactation cycle in a large population? Draw a histogram.

(b) What is the average lactation cycle of cows in a large population?

9. A firm operates in an uncertain profit environment. The firm takes an operating loss of one unit in a bad year, it makes a operating profit of two units in an average year, and it makes an operating profit of four units in a good year. At the beginning of a bad year, the firm may elect to shut down, avoiding the operating loss. Although the firm faces no fixed costs or shut-down costs, it incurs a start-up cost 0.2 units if it reopens after one or more periods of inactivity. The profit environment follows a stationary first-order Markov process with transition probabilities:

		to		
		bad	avg	good
	bad	0.4	0.5	0.1
from	avg	0.3	0.4	0.3
	good	0.1	0.5	0.4

(a) Suppose the firm adopts the policy of staying open regardless of the profit environment in any given year. Given that this is a bad year, how much profit can the firm expect to make one year from now, two years from now, three years from now, ten years from now?

(b) Suppose the firm adopts the following policy: (i) in a bad year, do not operate; (ii) in a good year, operate; and (iii) in an average year, do what you did the preceding year. Given that this is a bad year, how much profit can the firm expect to make one year from now, two years from now, three years from now?

Graph the expected profits for both parts on the same figure.

10. Prove theoretically that Gauss-Jacobi iteration applied to the linear equation $Ax = b$ must converge if A is diagonally dominant. You will need to use the Contraction mapping theorem and the result that $||My|| \leq ||M|| \cdot ||y||$ for any square matrix M and conformable vector y.

Chapter 3

Nonlinear Equations and Complementarity Problems

One of the most basic numerical operations encountered in computational economics is to find the solution of a system of non-linear equations. Nonlinear equations generally arise in one of two forms. In the nonlinear *rootfinding problem*, a function f from $\Re^n$ to $\Re^n$ is given and one must compute an n-vector x, called a *root* of f, that satisfies

$$f(x) = 0.$$

In the nonlinear *fixed-point problem*, a function g from $\Re^n$ to $\Re^n$ is given and one must compute an n-vector x called a *fixed-point* of g, that satisfies

$$g(x) = x.$$

The two forms are equivalent. The rootfinding problem may be recast as a fixed-point problem by letting $g(x) = x - f(x)$; conversely, the fixed-point problem may be recast as a rootfinding problem by letting $f(x) = x - g(x)$.

In the related *complementarity problem*, two n-vectors a and b, with $a < b$, and a function f from $\Re^n$ to $\Re^n$ are given, and one must compute an n-vector $x \in [a, b]$, that satisfies

$$\begin{aligned} x_i > a_i \;&\Rightarrow\; f_i(x) \geq 0 \qquad \forall i = 1, \ldots, n \\ x_i < b_i \;&\Rightarrow\; f_i(x) \leq 0 \qquad \forall i = 1, \ldots, n. \end{aligned}$$

The rootfinding problem is a special case of complementarity problem in which $a_i = -\infty$ and $b_i = +\infty$ for all i. However, the complementarity

problem is not simply to find a root that lies within specified bounds. An element $f_i(x)$ may be nonzero at a solution of the complementarity problem, provided that x_i equals one of the bounds a_i or b_i.

Nonlinear equations and complementarity problems arise directly in many economic applications. For example, the typical economic equilibrium model characterizes market prices and quantities with an equal number of supply, demand, and market clearing equations. If one or more of the equations is nonlinear, a nonlinear rootfinding problem arises. If the model is generalized to include constraints on prices and quantities arising from price supports, quotas, nonnegativity conditions, or limited production capacities, a nonlinear complementarity problem arises.

One also encounters nonlinear rootfinding and complementarity problems indirectly when maximizing or minimizing a real-valued function. An unconstrained optimum may be characterized by the condition that the first derivative of the function is zero—a rootfinding problem. A constrained optimum may be characterized by the Karush-Kuhn-Tucker conditions—a complementarity problem. Nonlinear equations and complementarity problems also arise as elementary tasks in solution procedures designed to solve more complicated functional equations. For example, the Euler functional equation of a dynamic optimization problem might be solved using a collocation method, which gives rise to a nonlinear equation or complementarity problem, depending on whether the actions are unconstrained or constrained, respectively.

Various practical difficulties arise with nonlinear equations and complementarity problems. In many applications, it is not possible to solve the nonlinear problem analytically. In these instances, the solution is often computed numerically using an iterative method that reduces the nonlinear problem to a sequence of linear problems. Such methods can be very sensitive to initial conditions and inherit many of the potential problems of linear equation methods, most notably rounding error and ill-conditioning. Nonlinear problems also present the added difficulty that they may have more than one solution.

Over the years, numerical analysts have studied nonlinear equations and complementarity problems extensively and have devised a variety of algorithms for solving them quickly and accurately. In many applications, one may use simple derivative-free methods, such as function iteration, which is applicable to fixed-point problems, or the bisection method, which is applicable to univariate rootfinding problems. In many applications, however, one

must rely on more sophisticated Newton and quasi-Newton methods, which use derivatives or derivative estimates to help locate the root or fixed-point of a function. These methods can be extended to complementarity problems using semismooth approximation methods.

3.1 Bisection Method

The *bisection method* is perhaps the simplest and most robust method for computing the root of a continuous real-valued function defined on a bounded interval of the real line. The bisection method is based on the Intermediate Value Theorem, which asserts that if a continuous real-valued function defined on an interval assumes two distinct values, then it must assume all values in between. In particular, if f is continuous, and $f(a)$ and $f(b)$ have different signs, then f must have at least one root x in $[a, b]$.

The bisection method is an iterative procedure. Each iteration begins with an interval known to contain or to 'bracket' a root of f, meaning the function has different signs at the interval endpoints. The interval is bisected into two subintervals of equal length. One of the two subintervals must have endpoints of different signs and thus must contain a root of f. This subinterval is taken as the new interval with which to begin the subsequent iteration. In this manner, a sequence of intervals is generated, each half the width of the preceding one, and each known to contain a root of f. The process continues until the width of the bracketing interval shrinks below an acceptable convergence tolerance.

The bisection method's greatest strength is its robustness. In contrast to other rootfinding methods, the bisection method is guaranteed to compute a root to a prescribed tolerance in a known number of iterations, provided valid data are input. Specifically, the method computes a root to a precision τ in no more than in $\log((b-a)/\tau)/\log(2)$ iterations. The bisection method, however, is applicable only to one-dimensional rootfinding problems and typically requires more iterations than other rootfinding methods to compute a root to a given precision, largely because it ignores information about the function's curvature. Given its relative strengths and weaknesses, the bisection method is often used in conjunction with other rootfinding methods. In this context, the bisection method is first used to obtain a crude approximation for the root. This approximation then becomes the starting point for a more precise rootfinding method that is used to compute a sharper, final

approximation to the root.

The following Matlab script computes the root of a user-supplied univariate function `f` using the bisection method. The user specifies two points at which `f` has different signs, `a` and `b`, and a convergence tolerance `tol`. The script makes use of the intrinsic Matlab function `sign`, which returns -1, 0, or 1 if its argument is negative, zero, or positive, respectively:

```
s = sign(f(a));
x = (a+b)/2;
d = (b-a)/2;
while d>tol;
   d = d/2;
   if s == sign(f(x))
      x = x+d;
   else
      x = x-d;
   end
end
```

In this implementation of the bisection algorithm, `d` begins each iteration equal to the distance from the current root estimate `x` to the boundaries of the bracketing interval. The value of `d` is cut in half, and the iterate is updated by increasing or decreasing its value by this amount, depending on the sign of `f(x)`. If $f(x)$ and $f(a)$ have the same sign, then the current x implicitly becomes the new left endpoint of the bracketing interval and x is moved d units toward b. Otherwise, the current x implicitly becomes the new right endpoint of the bracketing interval and x moved d units toward a.

The Matlab toolbox accompanying the textbook includes a function `bisect` that computes the root of a univariate function using the bisection method. The following script demonstrates how `bisect` may be used to compute the cube root of 2, or, equivalently, the root of the function $f(x) = x^3 - 2$:

```
f = inline('x^3-2');
x = bisect(f,1,2)
```

Execution of this script produces the result `x = 1.2599`. In this example, the initial bracketing interval is set to $[1, 2]$ and the root is computed to the default tolerance of $1.5 \cdot 10^{-8}$, or eight decimal places. The sequence of iterates is illustrated in Figure 3.1. The subroutine `bisect` is extensible in that it allows the user to override the default tolerance and to pass additional arguments for the function f; the subroutine also checks for input errors. The Matlab operation `inline` is used here to define the function whose root is sought.

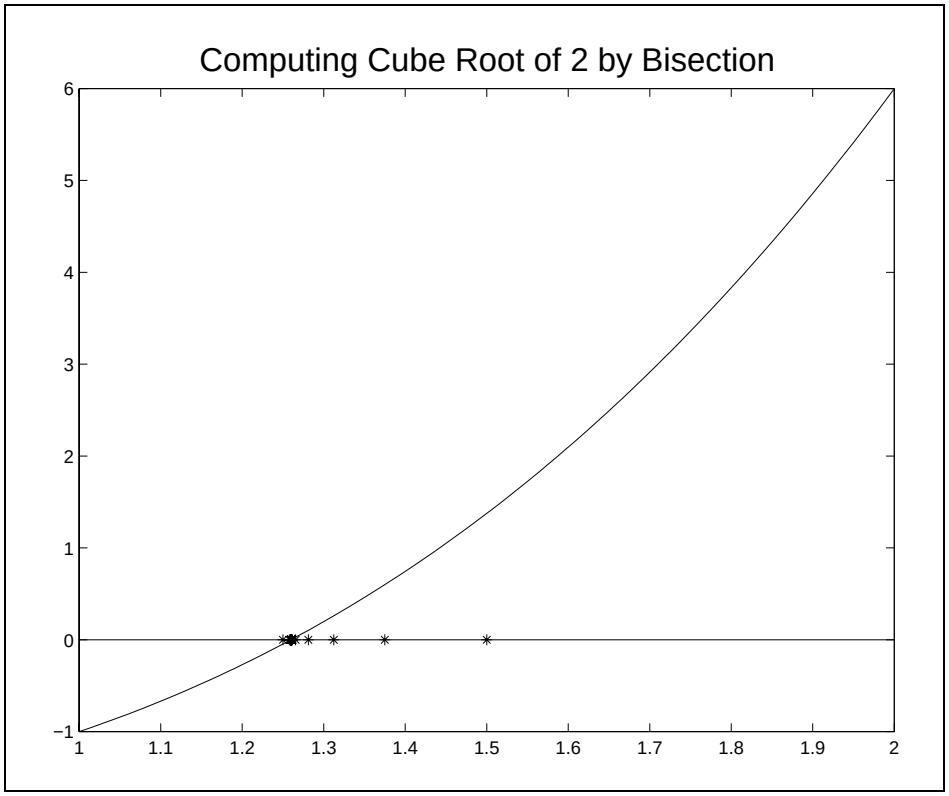

Figure 3.1

3.2 Function Iteration

Function iteration is a relatively simple technique that may be used to compute a fixed-point, $g(x) = x$, of a function from $\Re^n$ to $\Re^n$. The technique is also applicable to a rootfinding problem $f(x) = 0$, by recasting it as the equivalent fixed-point problem $g(x) = x - f(x) = x$.

Function iteration begins with the analyst supplying a guess $x^{(0)}$ for the fixed-point of g. Subsequent iterates are generated using the simple iteration rule

$$x^{(k+1)} \leftarrow g(x^{(k)}).$$

Since g is continuous, if the iterates converge, they converge to a fixed-point of g.

In theory, function iteration is guaranteed to converge to a fixed-point of g if g is differentiable and if the initial value of x supplied by the analyst is "sufficiently" close to a fixed-point x^* of g at which $\|g'(x^*)\| < 1$. Function iteration, however, often converges even when the sufficiency conditions are not met. Given that the method is relatively easy to implement, it is often worth trying before attempting to use more robust, but ultimately more complex methods, such as the Newton and quasi-Newton methods that are discussed in the following sections.

Computation of the fixed point of a univariate function $g(x)$ using function iteration is graphically illustrated in Figure 3.2. In this example, g possesses an unique fixed-point x^*, which is graphically characterized by the intersection of g and the 45-degree line. The algorithm begins with the analyst supplying a guess $x^{(0)}$ for the fixed-point of g. The next iterate $x^{(1)}$ is obtained by projecting upwards to the g function and then rightward to the 45-degree line. Subsequent iterates are obtained by repeating the projection sequence, tracing out a step function. The process continues until the iterates converge.

The Matlab toolbox accompanying the textbook includes a function `fixpoint` that computes the fixed-point of a multivariate function using function iteration. The following script computes the fixed point $x^* = 1$ of $g(x) = x^{0.5}$ to a default tolerance of $1.5 \cdot 10^{-8}$ starting from the initial guess $x = 0.4$:

```
g = inline('x^0.5');
x = fixpoint(g,0.4)
```

The subroutine `fixpoint` is extensible in that it allows the user to override the default tolerance and to pass additional arguments for the function g.

3.3 Newton's Method

In practice, most nonlinear rootfinding problems are solved using *Newton's method* or one of its variants. Newton's method is based on the principle of *successive linearization*. Successive linearization calls for a hard nonlinear problem to be replaced with a sequence of simpler linear problems whose solutions converge to the solution of the nonlinear problem. Newton's method is typically formulated as a rootfinding technique, but may be used to solve a fixed-point problem $g(x) = x$ by recasting it as the rootfinding problem $f(x) = x - g(x) = 0$.

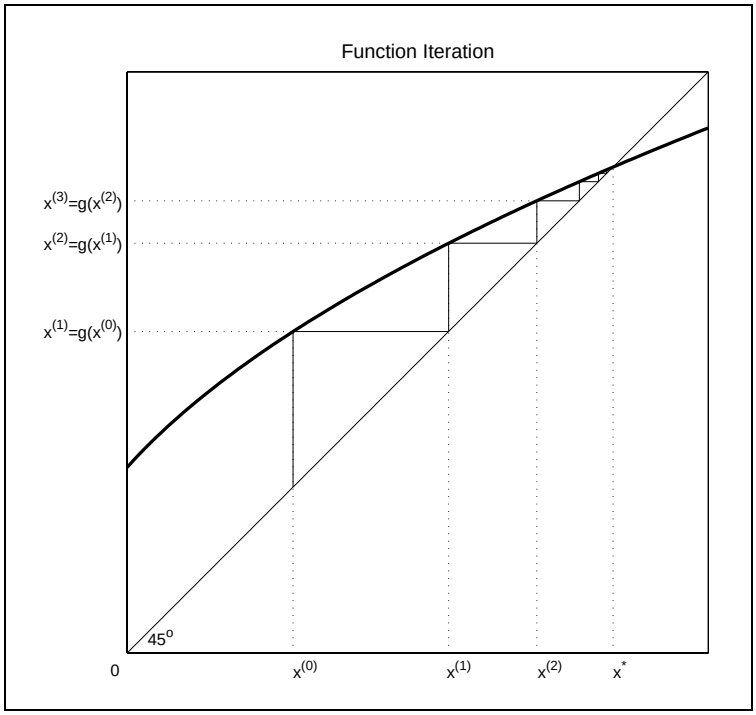

Figure 3.2

The univariate Newton method is graphically illustrated in Figure 3.3. The algorithm begins with the analyst supplying a guess $x^{(0)}$ for the root of f. The function f is approximated by its first-order Taylor series expansion about $x^{(0)}$, which is graphically represented by the line tangent to f at $x^{(0)}$. The root $x^{(1)}$ of the tangent line is then accepted as an improved estimate for the root of f. The step is repeated, with the root $x^{(2)}$ of the line tangent to f at $x^{(1)}$ taken as an improved estimate for the root of f, and so on. The process continues until the roots of the tangent lines converge.

More generally, the multivariate Newton method begins with the analyst supplying a guess $x^{(0)}$ for the root of f. Given $x^{(k)}$, the subsequent iterate $x^{(k+1)}$ is computed by solving the linear rootfinding problem obtained by replacing f with its first order Taylor approximation about $x^{(k)}$:

$$f(x) \approx f(x^{(k)}) + f'(x^{(k)})(x - x^{(k)}) = 0.$$

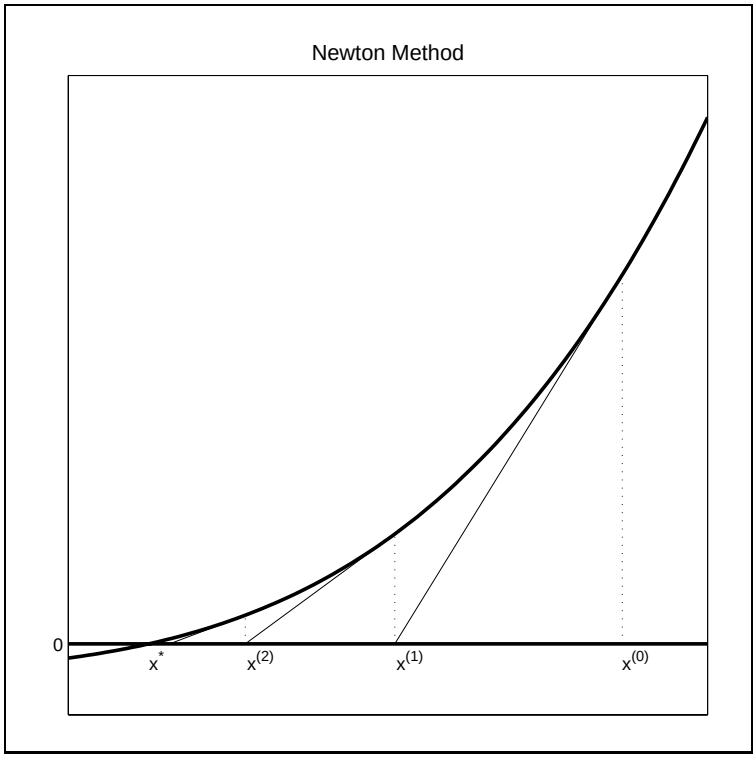

Figure 3.3

This yields the iteration rule

$$x^{(k+1)} \leftarrow x^{(k)} - [f'(x^{(k)})]^{-1} f(x^{(k)}).$$

The following Matlab script computes the root of a function f using Newton's method. It assumes that the user has provided an initial guess `x` for the root, a convergence tolerance `tol`, and an upper limit `maxit` on the number of iterations. It calls a user-supplied routine `f` that computes the value `fval` and Jacobian `fjac` of the function at an arbitrary point `x`. To conserve on storage, only the most recent iterate is stored:

```
for it=1:maxit
   [fval,fjac] = f(x);
   x = x - fjac\fval;
   if norm(fval) < tol, break, end
end
```

In theory, Newton's method converges if f is continuously differentiable and if the initial value of x supplied by the analyst is "sufficiently" close to a root of f at which f' is invertible. There is, however, no generally practical formula for determining what sufficiently close is. Typically, an analyst makes a reasonable guess for the root f and counts his blessings if the iterates converge. If the iterates do not converge, then the analyst must look more closely at the properties of f to find a better starting value, or change to another rootfinding method. Newton's method can be robust to the starting value if f is well behaved, for example, if f has monotone derivatives. Newton's method can be very sensitive to starting value, however, if the function behaves erratically, for example, if f has high derivatives that change sign frequently. Finally, in practice it is not sufficient for f' to be merely invertible at the root. If f' is invertible but ill-conditioned, then rounding errors in the vicinity of the root can make it difficult to compute a precise approximation to the root using Newton's method.

The Matlab toolbox accompanying the textbook includes a function `newton` that computes the root of a function using the Newton's method. To illustrate the use of this function, consider a simple Cournot duopoly model, in which the inverse demand for a good is

$$p = P(q) = q^{-1/\eta}$$

and the two firms producing the good face cost functions

$$C_i(q_i) = \tfrac{1}{2} c_i q_i^2, \text{ for } i = 1, 2.$$

The profit for firm i is

$$\pi_i(q_1, q_2) = P(q_1 + q_2) q_i - C(q_i).$$

If firm i takes the other's firms output as given, it will choose its output level so as to solve

$$\partial \pi_i / \partial q_i = P(q_1 + q_2) + P'(q_1 + q_2) q_i - C_i'(q_i) = 0.$$

Thus, the market equilibrium outputs, q_1 and q_2, are the roots of the two nonlinear equations

$$f_i(q) = (q_1 + q_2)^{-1/\eta} - (1/\eta)(q_1 + q_2)^{-1/\eta - 1} q_i - c_i q_i = 0, \text{ for } i = 1, 2.$$

Suppose one wished to use the function `newton` to compute for the market equilibrium quantities, assuming $\eta = 1.6$, $c_1 = 0.6$ and $c_2 = 0.8$. The first step would be write a Matlab function that gives the value and Jacobian of f at arbitrary vector of quantities q:

```
function [fval,fjac] = cournot(q)
c = [0.6; 0.8]; eta = 1.6; e = -1/eta;
fval = sum(q)^e + e*sum(q)^(e-1)*q - diag(c)*q;
fjac = e*sum(q)^(e-1)*ones(2,2) + e*sum(q)^(e-1)*eye(2) ...
     + (e-1)*e*sum(q)^(e-2)*q*[1 1] - diag(c);
```

Making an initial guess of, say $q_1 = q_2 = 0.2$, a call to `newton`

```
q = newton(f,[0.2;0.2]);
```

will compute the equilibrium quantities $q_1 = 0.8396$ and $q_2 = 0.6888$ to the default tolerance of $1.5 \cdot 10^{-8}$. The subroutine `newton` is extensible in that it allows the user to override the default tolerance and limit on the number of iterations, and allows the user to pass additional arguments for the function f, if necessary.

The path taken by `newton` to the Cournot equilibrium solution from an initial guess of $(0.2, 0.2)$ is illustrated by the dashed line in Figure 3.4. Here, the Cournot market equilibrium is the intersection of the zero contours of f_1 and f_2, which may be interpreted as the reaction functions for the two firms. In this case Newton's method works very well, needing only a few steps to effectively land on the root.

3.4 Quasi-Newton Methods

Quasi-Newton methods offer an alternative to Newton's method for solving rootfinding problems. Quasi-Newton methods are based on the same successive linearization principle as Newton's method, except that they replace the Jacobian f' with an estimate that is easier to compute. Quasi-Newton methods are easier to implement and less likely to fail due to programming errors than Newton's method because the analyst need not explicitly code the derivative expressions. Quasi-Newton methods, however, often converge more slowly than Newton's method and additionally require the analyst to supply an initial estimate of the function's Jacobian.

The *secant method* is the most widely used univariate quasi-Newton method. The secant method is identical to the univariate Newton method, except that it replaces the derivative of f with a finite-difference approximation constructed from the function values at the two previous iterates:

$$f'(x^{(k)}) \approx \frac{f(x^{(k)}) - f(x^{(k-1)})}{x^{(k)} - x^{(k-1)}}.$$

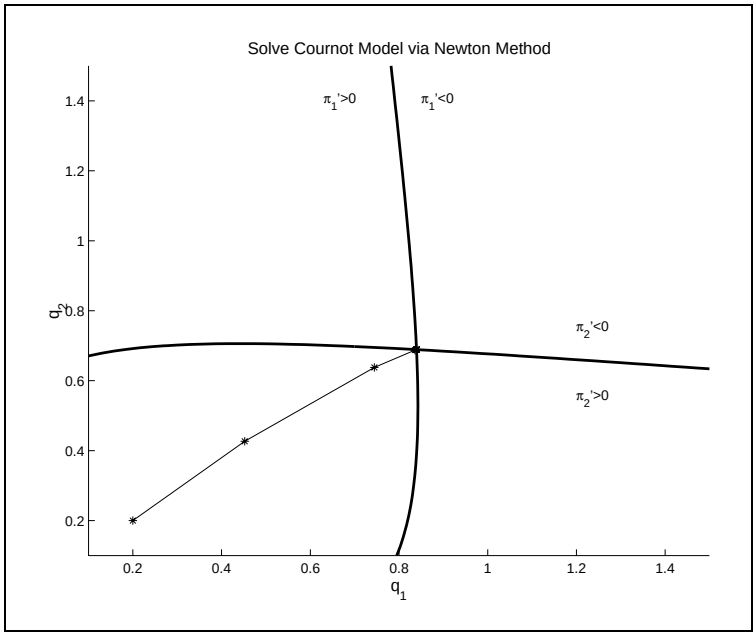

Figure 3.4

This yields the iteration rule

$$x^{(k+1)} \leftarrow x^{(k)} - \frac{x^{(k)} - x^{(k-1)}}{f(x^{(k)}) - f(x^{(k-1)})} f(x^{(k)}).$$

Unlike the Newton method, the secant method requires two, rather than one starting value.

The secant method is graphically illustrated in Figure 3.5. The algorithm begins with the analyst supplying two distinct guesses $x^{(0)}$ and $x^{(1)}$ for the root of f. The function f is approximated using the secant line passing through $x^{(0)}$ and $x^{(1)}$, whose root $x^{(2)}$ is accepted as an improved estimate for the root of f. The step is repeated, with the root $x^{(3)}$ of the secant line passing through $x^{(1)}$ and $x^{(2)}$ taken as an improved estimate for the root of f, and so on. The process continues until the roots of the secant lines converge.

Broyden's method is the most popular multivariate generalization of the univariate secant method. Broyden's method generates a sequence of vectors $x^{(k)}$ and matrices $A^{(k)}$ that approximate the root of f and the Jacobian f' at the root, respectively. Broyden's method begins with the analyst supplying

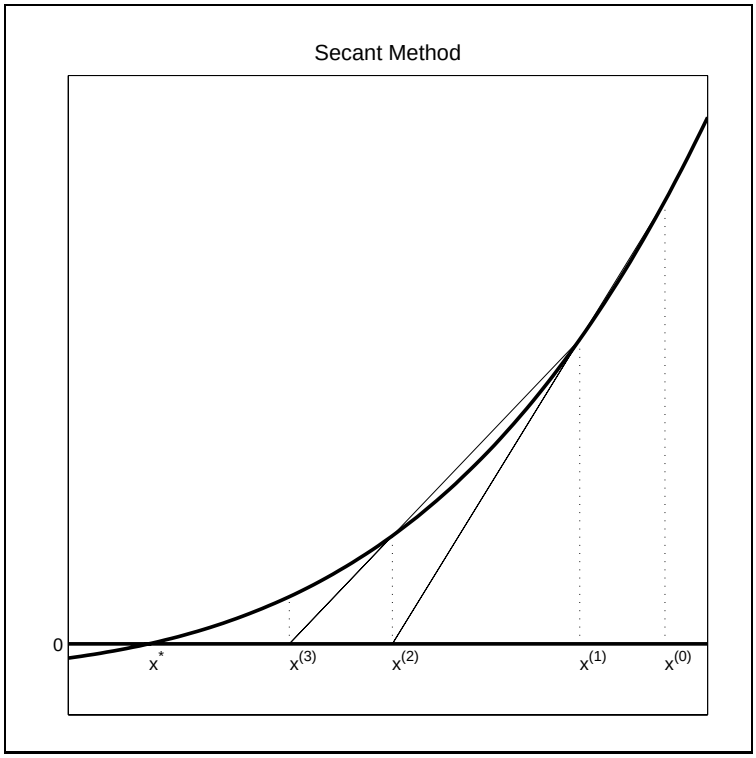

Figure 3.5

a guess $x^{(0)}$ for the root of the function and a guess $A^{(0)}$ for the Jacobian of the function at the root. Often, $A^{(0)}$ is set equal to the numerical Jacobian of f at $x^{(0)}$.[1] Alternatively, some analysts use a rescaled identity matrix for $A^{(0)}$, though this typically will require more iterations to obtain a solution than if a numerical Jacobian is computed at the outset.

Given $x^{(k)}$ and $A^{(k)}$, one updates the root approximation by solving the linear rootfinding problem obtained by replacing f with its first-order Taylor approximation about $x^{(k)}$:

$$f(x) \approx f(x^{(k)}) + A^{(k)}(x - x^{(k)}) = 0.$$

This yields the root approximation iteration rule

$$x^{(k+1)} \leftarrow x^{(k)} - (A^{(k)})^{-1} f(x^{(k)}).$$

[1]Numerical differentiation is discussed in Chapter 5.

Broyden's method then updates the Jacobian approximant $A^{(k)}$ by making the smallest possible change, measured in the Frobenius matrix norm, that is consistent with the *secant condition*, which any reasonable Jacobian estimate should satisfy to a first order:

$$f(x^{(k+1)}) - f(x^{(k)}) = A^{(k+1)}(x^{(k+1)} - x^{(k)}).$$

This yields the iteration rule

$$A^{(k+1)} \leftarrow A^{(k)} + \frac{f(x^{(k+1)})\delta^{(k)\top}}{\delta^{(k)\top}\delta^{(k)}}$$

where $\delta^{(k)} = x^{(k+1)} - x^{(k)}$.

In practice, Broyden's method may be accelerated by avoiding the linear solve. This can be accomplished by retaining and updating the Broyden estimate of the inverse of the Jacobian, rather than that of the Jacobian itself. Broyden's method with inverse update generates a sequence of vectors $x^{(k)}$ and matrices $B^{(k)}$ that approximate the root of f and the inverse Jacobian f'^{-1} at the root, respectively. It uses the root approximation iteration rule

$$x^{(k+1)} \leftarrow x^{(k)} - B^{(k)} f(x^{(k)})$$

and inverse update rule

$$B^{(k+1)} \leftarrow B^{(k)} + ((\delta^{(k)} - \gamma^{(k)})dx^{\top} B^{(k)})/(\delta^{(k)\top}\gamma^{(k)})$$

where $\gamma^{(k)} = B^{(k)}(f(x^{(k+1)}) - f(x^{(k)}))$. Most implementations of Broyden's methods employ the inverse update rule because of its modest speed advantage over Broyden's method with Jacobian update.

In theory, Broyden's method converges if f is continuously differentiable, if $x^{(0)}$ is "sufficiently" close to a root of f at which f' is invertible, and if $A^{(0)}$ or $B^{(0)}$ are "sufficiently" close to the Jacobian or inverse Jacobian of f at that root. There is, however, no generally practical formula for determining what sufficiently close is. Like Newton's method, the robustness of Broyden's method depends on the regularity of f and its derivatives. Broyden's method may also have difficulty computing a precise root estimate if f' is ill-conditioned near the root. It is important to also note that the sequence approximants $A^{(k)}$ and $B^{(k)}$ need not, and typically do not, converge to the Jacobian and inverse Jacobian of f at the root, respectively, even if the $x^{(k)}$ converge to a root of f.

The following Matlab script computes the root of a user-supplied multivariate function f using Broyden's method with inverse update. The script assumes that the user has written a Matlab routine `f` that evaluates the function at an arbitrary point and that the user has specified a starting point `x`, a convergence tolerance `tol`, and a limit on the number of iterations `maxit`. The script also computes an initial guess for the inverse Jacobian by inverting the finite difference derivative computed using the toolbox function `fdjac`, which is discussed in a later section.

```
fjacinv = inv(fdjac(f,x));
fval = f(x);
for it=1:maxit
   fnorm = norm(fval);
   if fnorm<tol, break; end
   dx = -fjacinv*fval;
   x = x+dx;
   fold = fval;
   fval = f(x);
   temp = fjacinv*(fval-fold);
   fjacinv = fjacinv + ((dx-temp)*dx'*fjacinv)/(dx'*temp);
end
```

The Matlab toolbox accompanying the textbook includes a function `broyden` that computes the root of a function using Broyden's method with inverse update. To illustrate the use of this function, consider the simple Cournot duopoly model, introduced in the preceding subsection. The first step in solving the model using Broyden's method would be to write a Matlab function that gives the value of f at arbitrary vector of quantities q:

```
function fval = f(q)
c = [0.6; 0.8]; eta = 1.6; e = -1/eta;
fval = sum(q)^e + e*sum(q)^(e-1)*q - diag(c)*q;
```

Note that the function need not return the Jacobian of f because Broyden method does not require it. Making an initial guess of, say $q_1 = q_2 = 0.2$, a call to `broyden`

```
q = broyden(f,[0.2;0.2]);
```

will compute the equilibrium quantities $q_1 = 0.8396$ and $q_2 = 0.6888$ to the default tolerance of $1.5 \cdot 10^{-8}$. The subroutine `broyden` is extensible in that it allows the user to enter an initial estimate of the Jacobian estimate, if available, and allows the user to override the default tolerance and limit

on the number of iterations. The subroutine also allows the user to pass additional arguments for the function f, if necessary.

The path taken by `broyden` to the Cournot equilibrium solution from an initial guess of $(0.2, 0.2)$ is illustrated by the dashed line in Figure 3.6. In this case Broyden's method works well and not altogether very different from Newton's method. However, a close comparison of Figures 3.4 and 3.6 demonstrates that Broyden's method takes more iterations and follows a somewhat more circuitous route than Newton's method.

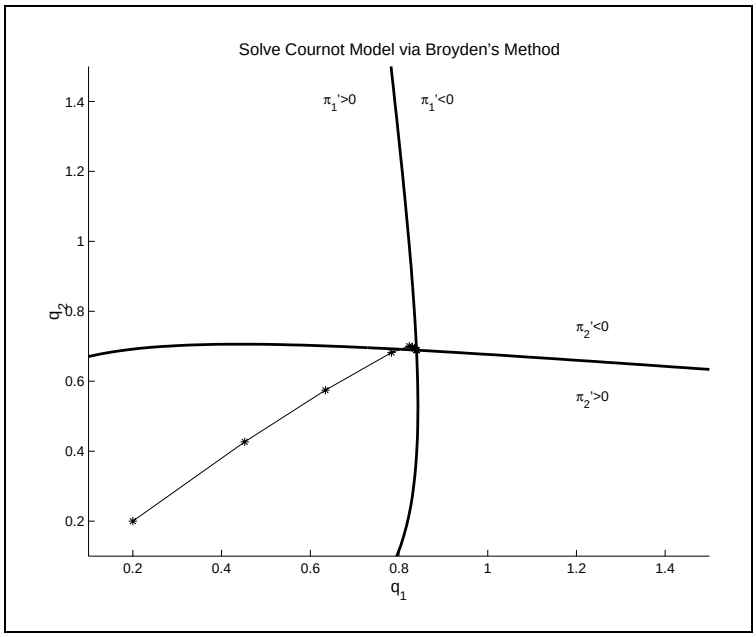

Figure 3.6

3.5 Problems With Newton Methods

There are no fail-proof methods for solving multivariate non-linear equations. Several difficulties commonly arise in the application of Newton and quasi-Newton methods. The most common cause of failure of Newton-type methods is coding errors committed by the analyst. The next most common cause of failure is the specification of a starting point that is not sufficiently

close to a root. And yet another common cause of failure is an ill-conditioned Jacobian at the root. All of these problems can often be mitigated by appropriate action, though they cannot always be eliminated altogether.

There first cause of failure, coding error, may seem obvious and not specific to rootfinding problems. It must be emphasized, however, that with Newton's method, the likelihood of committing an error in coding the analytic Jacobian of the function is often high. A careful analyst can avoid Jacobian coding errors in two ways. First, the analyst could use Broyden's method instead of Newton's method to solve the rootfinding problem. Broyden's method is derivative-free and does not require the explicit coding of the function's analytic Jacobian. Second, the analyst can perform a simple, but highly effective check of his code by comparing the values computed by his analytic derivatives to those computed using finite difference methods. Such a check will almost always detect an error in either the code that returns the function's value or the code that returns its Jacobian.

A comparison of analytic and finite difference derivatives can easily be performed using the `checkjac` routine provided with the Matlab toolbox accompanying this textbook. This function computes the analytic and finite difference derivatives of a function at a specified evaluation point and returns the index and magnitude of the largest deviation. The function may be called as follows:

```
[error,i,j] = checkjac(f,x)
```

Here, we assume that the user has coded a Matlab function f that returns the function value and analytic derivatives at a specified evaluation point x. Execution returns `error`, the highest absolute difference between an analytic and finite difference cross-partial derivative of f, and its index `i` and `j`. A large deviation indicates that the either the i, j^{th} partial derivative or the i^{th} function value may be incorrectly coded.

The second problem, a poor starting value, can be partially addressed by 'backstepping'. If taking a full Newton (or quasi-Newton) step $x + dx$ does not offer an improvement over the current iterate x, then one 'backsteps' toward the current iterate x by repeatedly cutting dx in half until $x + dx$ does offer an improvement. Whether a step dx offers an improvement is measured by the Euclidean norm $\|f(x)\| = \frac{1}{2} f(x)^\top f(x)$. Clearly, $\|f(x)\|$ is precisely zero at a root of f, and is positive elsewhere. Thus, one may view an iterate as yielding an improvement over the previous iterate if it reduces the function norm, that is, if $\|f(x)\| > \|f(x + dx)\|$. Backstepping prevents

Newton and quasi-Newton methods from taking a large step in the wrong direction, substantially improving their robustness.

Backstepping, however, has the tendency in some applications to begin taking very short steps as the iterations approach the root. One way to prevent this from happening is to employ a 'safeguarding' strategy. A simple safeguarding strategy is to seek a reduction in the function norm by repeatedly cutting the Newton step in half, but to stop if the norm begins to rise. This prevents the procedure from getting stuck near the root. The maximum number of allowable backsteps should be no more than, say, 30, which implies a minimum step size that is 2^{-30} or approximately $1e-9$ times the Newton step.

The following Matlab script computes the root of a function using a safeguarded Newton's method. It assumes that the user has specified a maximum number `maxit` of Newton iterations, a maximum number `maxsteps` of backstep iterations, and a convergence tolerance `tol`, along with the name of the function `f` and an initial value `x`:

```
for it=1:maxit
   [fval,fjac] = f(x);
   fnorm = norm(fval);
   if fnorm<tol, return, end
   dx = -fjac\fval;
   fnormold = inf;
   for backstep=1:maxsteps
      xnew = x + dx;
      fnew = f(x);
      fnormnew = norm(fnew);
      if fnormold<fnormnew | fnormnew<fnorm, break, end
      fnormold = fnormnew;
      dx = dx/2;
   end
   x = x+dx;
end
```

Safeguarded backstepping may also implemented with Broyden's method, except that the Jacobian updating procedure must be modified to ensure that the secant condition is satisfied. The `newton` and `broyden` routines supplied with the Matlab toolbox accompanying the textbook both employ safeguarded backstepping.

The third problem, an ill-conditioned Jacobian at the root, occurs less often, but should not be ignored. An ill-conditioned Jacobian can render inaccurately computed Newton step dx, creating severe difficulties for the conver-

gence of Newton and Newton-type methods. In some cases, ill-conditioning is a structural feature of the underlying model and cannot be eliminated. However, in many cases, ill-conditioning is inadvertently and unnecessarily introduced by the analyst. A common source of avoidable ill-conditioning arises when the natural units of measurements for model variables yield values that vary vastly in order of magnitude. When this occurs, the analyst should consider rescaling the variables so that their values have comparable orders of magnitude, preferably close to unity. Rescaling will generally lead to faster execution time and more accurate results.

3.6 Choosing a Solution Method

Numerical analysts have special terms that they use to classify the rates at which iterative routines converge. Specifically, a sequence of iterates $x^{(k)}$ is said to converge to x^* at a rate of order p if there is constant $C > 0$ such that

$$\|x^{(k+1)} - x^*\| \leq C\|x^{(k)} - x^*\|^p$$

for sufficiently large k. In particular, the rate of convergence is said to be *linear* if $C < 1$ and $p = 1$, *superlinear* if $1 < p < 2$, and *quadratic* if $p = 2$.

The asymptotic rates of convergence of the nonlinear equation solution methods discussed earlier are well known. The bisection method converges at a linear rate with $C = 1/2$. The function iteration method converges at a linear rate with C equal to $\|f'(x^*)\|$. The secant and Broyden methods converge at a superlinear rate, with $p \approx 1.62$. And Newton's method converges at a quadratic rate. The rates of convergence are asymptotically valid, provided that the algorithms are given "good" initial data.

Consider a simple example. The function $g(x) = \sqrt{x}$ has an unique fixed-point $x^* = 1$. Function iteration may be used to compute the fixed-point. One can also compute the fixed-point by applying Newton's method or the secant method to the equivalent rootfinding problem $f(x) = x - \sqrt{x} = 0$.

Starting from $x^{(0)} = 0.5$, and using a finite difference derivative for the first secant method iteration, the approximation error $|x^{(k)} - x^*|$ produced by the three methods are:

k	Function Iteration	Broyden's Method	Newton's Method
1	2.9e-001	-2.1e-001	-2.1e-001
2	1.6e-001	3.6e-002	-8.1e-003
3	8.3e-002	1.7e-003	-1.6e-005
4	4.2e-002	-1.5e-005	-6.7e-011
5	2.1e-002	6.3e-009	0.0e+000
6	1.1e-002	2.4e-014	0.0e+000
7	5.4e-003	0.0e+000	0.0e+000
8	2.7e-003	0.0e+000	0.0e+000
9	1.4e-003	0.0e+000	0.0e+000
10	6.8e-004	0.0e+000	0.0e+000
15	2.1e-005	0.0e+000	0.0e+000
20	6.6e-007	0.0e+000	0.0e+000
25	2.1e-008	0.0e+000	0.0e+000

This simple experiment generates convergence patterns that are typical for the various iterative nonlinear equation solution algorithms used in practice. Newton's method converges in fewer iterations than the quasi-Newton method, which in turn converges in fewer iterations than function iteration. Both the Newton and quasi-Newton methods converge to machine precision very quickly, in this case 5 or 6 iterations. As the iterates approach the solution, the number of significant digits in the Newton and quasi-Newton approximants begin to double with each iteration.

However, the rate of convergence, measured in number of iterations, is only one determinant of the computational efficiency of a solution algorithm. Algorithms differ in the number of arithmetic operations, and thus the computational effort required per iteration. For multivariate problems, function iteration requires only a function evaluation; Broyden's method with inverse update requires a function evaluation and a matrix-vector multiplication; and Newton's method requires a function evaluation, a derivative evaluation, and the solution of a linear equation. In practice, function iteration tends to require the most overall computational effort to achieve a given accuracy than the other two methods. However, whether Newton's method or Broyden's method requires the most overall computational effort to achieve convergence in a given application depends largely on the dimension of x and complexity of the derivative. Broyden's method will tend to be computationally more

efficient than Newton's method if the derivative is costly to evaluate.

An important factor that must be considered when choosing a nonlinear equation solution method is developmental effort. Developmental effort is the effort exerted by the analyst to produce a viable, convergent computer code—this includes the effort to write the code, the effort to debug and verify the code, and the effort to find suitable starting values. Function iteration and quasi-Newton methods involve the least developmental effort because they do not require the analyst to correctly code the derivative expressions. Newton's method typically requires more developmental effort because it additionally requires the analyst to correctly code derivative expressions. The developmental cost of Newton's method can be quite high if the derivative matrix involves many complex or irregular expressions.

Experienced analysts use certain rules of thumb when selecting a nonlinear equation solution method. If the nonlinear equation is of small dimension, say univariate or bivariate, *or* the function derivatives follow a simple pattern and are relatively easy to code, then development costs will vary little among the different methods and computational efficiency should be the main concern, particularly if the equation is to be solved many times. In this instance, Newton's method is usually the best first choice.

If the nonlinear equation involves many complex or irregular function derivatives, or if the derivatives are expensive to compute, then the Newton's method it less attractive. In such instances, quasi-Newton and function iteration methods may make better choices, particularly if the nonlinear equation is to be solved very few times. If the nonlinear equation is to be solved many times, however, the faster convergence rate of Newton's method may make the development costs worth incurring.

3.7 Complementarity Problems

Many economic models naturally take the form of a complementary problem rather than a rootfinding or fixed point problem. In the complementarity problem, two n-vectors a and b, with $a < b$, and a function f from $\Re^n$ to $\Re^n$ are given, and one must find an n-vector $x \in [a, b]$, that satisfies

$$\begin{aligned} x_i > a_i \;&\Rightarrow\; f_i(x) \geq 0 \qquad \forall i = 1, \ldots, n \\ x_i < b_i \;&\Rightarrow\; f_i(x) \leq 0 \qquad \forall i = 1, \ldots, n. \end{aligned}$$

The complementarity conditions require that $f_i(x) = 0$ whenever $a_i < x_i < b_i$. The complementarity problem thus includes the rootfinding problem as a special case in which $a_i = -\infty$ and $b_i = +\infty$ for all i. The complementarity problem, however, is not to find a root that lies within specified bounds. An element $f_i(x)$ may be nonzero at a solution of a complementarity problem, though only if x_i equals one of its bounds. For the sake of brevity, we denote the complementarity problem $\mathrm{CP}(f, a, b)$.

Complementarity problems arise naturally in economic equilibrium models. In this context, x is an n-vector that represents the levels of certain economic activities. For each $i = 1, 2, \ldots, n$, a_i denotes a lower bound on activity i, b_i denotes an upper bound on activity i, and $f_i(x)$ denotes the marginal arbitrage profit associated with activity i. Disequilibrium arbitrage profit opportunities exist if either $x_i < b_i$ and $f_i(x) > 0$, in which case an incentive exists to increase x_i, or $x_i > a_i$ and $f_i(x) < 0$, in which case an incentive exists to decrease x_i. An arbitrage-free economic equilibrium obtains if and only if x solves the complementarity problem $\mathrm{CP}(f, a, b)$.

Complementarity problems also arise naturally in economic optimization models. Consider maximizing a function $F : \Re^n \mapsto \Re$ subject to the simple bound constraint $x \in [a, b]$. The Karush-Kuhn-Tucker theorem asserts that x solves the bounded maximization problem only if it solves the complementarity problem $\mathrm{CP}(f, a, b)$ where $f_i(x) = \partial F / \partial x_i$. Conversely, if F is strictly concave and x solves the complementarity problem $\mathrm{CP}(f, a, b)$, then x solves the bounded maximization problem.

As a simple example of a complementarity problem, consider the well-known Marshallian competitive price equilibrium model. In this model, competitive equilibrium obtains if and only if excess demand $E(p)$, the difference between quantity demanded and quantity supplied at price p, is zero. Suppose, however, that the government imposes a price ceiling $\bar{p}$ that it enforces through fiat or direct market intervention. It is then possible for excess demand to exist at equilibrium, but only if price ceiling is binding. In the presence of a price ceiling, the equilibrium market price is the solution to the complementarity problem $\mathrm{CP}(E, 0, \bar{p})$.

A more interesting example of a complementarity problem is the single commodity competitive spatial price equilibrium model. Suppose that there are n distinct regions and that excess demand for the commodity in region i is a function $E_i(p_i)$ of the price p_i in the region. In the absence of trade among regions, equilibrium is characterized by the condition that $E_i(p_i) = 0$ in each region i, a rootfinding problem. Suppose, however, that trade can take place

among regions, and that the cost of transporting one unit of the good from region i to region j is a constant c_{ij}. Denote by x_{ij} the amount of the good that is produced in region i and consumed in region j and suppose that this quantity cannot exceed a given shipping capacity b_{ij}. In this market, $p_j - p_i - c_{ij}$ is the unit arbitrage profit available from shipping one unit of the commodity from region i to region j. When the arbitrage profit is positive, an incentive exists to increase shipments; when the arbitrage profit is negative, an incentive exists to decrease shipments. Equilibrium obtains only if all spatial arbitrage profit opportunities have been eliminated. This requires that, for all pairs of regions i and j, $0 \leq x_{ij} \leq b_{ij}$ and

$$\begin{array}{lcl} x_{ij} > 0 & \Rightarrow & p_j - p_i - c_{ij} \geq 0 \\ x_{ij} < b_{ij} & \Rightarrow & p_j - p_i - c_{ij} \leq 0. \end{array}$$

To formulate the spatial price equilibrium model as a complementarity problem, note that market clearing requires that net imports equal excess demand in each region i:

$$\sum_k [x_{ki} - x_{ik}] = E_i(p_i).$$

This implies that

$$p_i = E_i^{-1}\left(\sum_k [x_{ki} - x_{ik}]\right).$$

If

$$f_{ij}(x) = E_j^{-1}\left(\sum^k [x_{kj} - x_{jk}]\right) - E_i^{-1}\left(\sum^k [x_{ki} - x_{ik}]\right) - c_{ij}$$

then x is a spatial equilibrium trade flow if and only if x solves the complementary problem $\mathrm{CP}(f, 0, b)$, where x, f and b are vectorized and written as n^2 by 1 vectors.

In order to understand the mathematical structure of the complementarity problem, it is instructive to consider the simplest case: the univariate linear complementarity problem. Figure 3.7a-c illustrate the three possible subcases when f is negatively sloped. In all three subcases, an unique equilibrium solution exists. In Figure 3.7a, $f(a) \leq 0$ and the unique equilibrium solution is $x^* = a$; in Figure 3.7b, $f(b) \geq 0$ and the unique equilibrium

solution is $x^* = b$; and in Figure 3.7c, $f(a) > 0 > f(b)$ and the unique equilibrium solution lies between a and b. In all three subcases, the equilibrium is stable in that the economic incentive at nearby disequilibrium points is to return to the equilibrium.

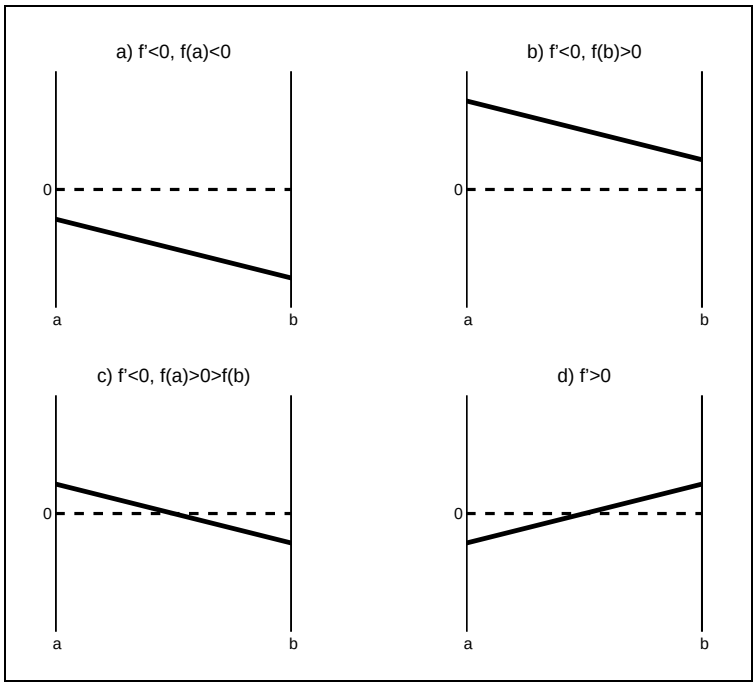

Figure 3.7

Figure 3.7d illustrates the difficulties that can arise when f is positively sloped. Here, multiple equilibrium solutions arise, one in the interior of the interval and one at each endpoint. The interior equilibrium, moreover, is unstable in that the economic incentive at nearby disequilibrium points is to move away from the interior equilibrium toward one of the corner equilibria.

More generally, multivariate complementarity problems are guaranteed to possess an unique solution if f is strictly negative monotone, that is, if $(x-y)'(f(x)-f(y)) < 0$ whenever $x, y \in [a, b]$ and $x \neq y$. This will be true for most well-posed economic equilibrium models. It will also be true when the complementarity problem derives from a bound constrained maximization problem in which the objective function is strictly concave.

3.8 Complementarity Methods

Although the complementarity problem appears quite different from the ordinary rootfinding problem, it actually can be reformulated as one. In particular, x solves the complementarity problem $\mathrm{CP}(f, a, b)$ if and only if it solves the rootfinding problem

$$\hat{f}(x) = \min(\max(f(x), a - x), b - x) = 0.$$

A formal proof of the equivalence between the complementarity problem $\mathrm{CP}(f, a, b)$ and its 'minmax' rootfinding formulation $\hat{f}(x) = 0$ is straightforward, but requires a somewhat tedious enumeration of several possible cases, which we leave as an exercise for the reader. The equivalence, however, can easily be demonstrated graphically for the univariate complementarity problem.

Figure 3.8 illustrates minmax rootfinding formulation of the same four univariate complementarity problems examined in Figure 3.7. In all four plots, the curves $y = a - x$ and $y = b - x$ are drawn with narrow dashed lines, the curve $y = f(x)$ is drawn with a narrow solid line, and the curve $y = \hat{f}(x)$ is drawn with a thick solid line; clearly, in all four figures, $\hat{f}$ lies between the lines $y = x - a$ and $y = x - b$ and coincides with f inside the lines. In Figure 3.8a, $f(a) \leq 0$ and the unique solution to the complementarity problem is $x^* = a$, which coincides with the unique root of $\hat{f}$; in Figure 3.8b, $f(b) \geq 0$ and the unique solution to the complementarity problem is $x^* = b$, which coincides with the unique root of $\hat{f}$; in Figure 3.8c, $f(a) > 0 > f(b)$ and the unique solution to the complementarity problem lies between a and b and coincides with the unique root of $\hat{f}$ (and f). In Figure 3.7d, f is upwardly sloped and possesses multiple roots, all of which, again, coincide with roots of $\hat{f}$.

The reformulation of the complementarity problem as a rootfinding problem suggests that it may be solved using standard rootfinding algorithms, such as Newton's method. To implement Newton's method for the minmax rootfinding formulation requires computation of the Jacobian $\hat{J}$ of $\hat{f}$. The i^{th} row of $\hat{J}$ may be derived directly from the Jacobian J of f:

$$\hat{J}_i(x) = \begin{cases} J_i(x), & \text{for } a_i - x_i < f_i(x) < b_i - x_i, \\ -I_i. & \text{otherwise} \end{cases}$$

Here, I_i is the i^{th} row of the identity matrix.

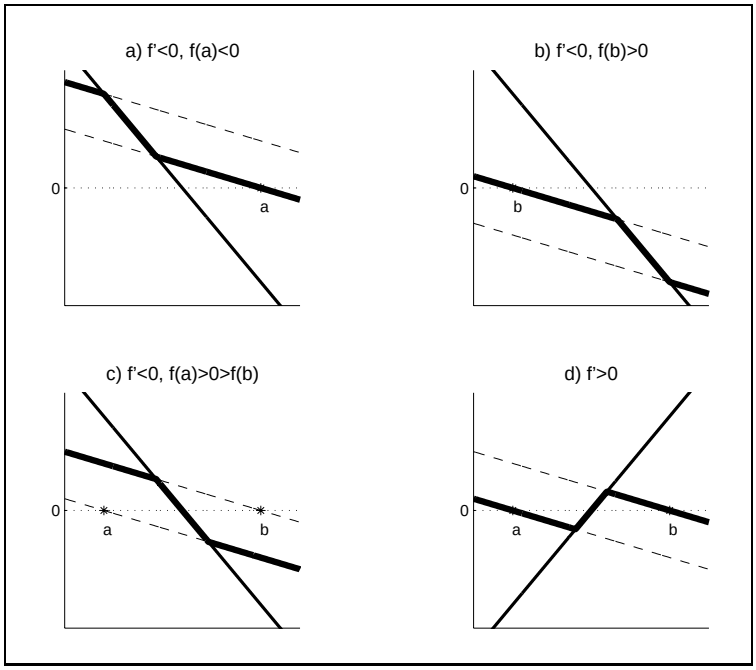

Figure 3.8

The following Matlab script computes the solution of the complementarity problem CP(f, a, b) by applying Newton's method to the equivalent minmax rootfinding formulation. The script assumes that the user has provided the lower and upper bounds `a` and `b`, a guess `x` for the solution of the complementarity problem, a convergence tolerance `tol`, and an upper limit `maxit` on the number of iterations. It calls a user-supplied routine `f` that computes the value `fval` and Jacobian `fjac` of the function at an arbitrary point `x`:

```
for it=1:maxit
   [fval,fjac] = f(x);
   fhatval = min(max(fval,a-x),b-x);
   fhatjac = -eye(length(x));
   i = find(fval>a-x & fval<b-x);
   fhatjac(i,:) = fjac(i,:);
   x = x - fhatjac\fhatval;
   if norm(fhatval)<tol, break, end
end
```

Using Newton's method to find a root of $\hat{f}$ will often work well. However, in many cases, the nondifferentiable kinks in $\hat{f}$ create difficulties for Newton's method, undermining its ability to converge rapidly. One way to deal with the kinks is to replace $\hat{f}$ with a function that has the same roots, but is smoother and therefore less prone to numerical difficulties. One function that has proven very effective for solving the complementarity problem in practical applications is Fischer's function

$$\tilde{f}(x) = \phi^+(\phi^-(f(x), x-a), x-b),$$

where

$$\phi^{\pm}(u,v) = u + v \pm \sqrt{u.*u + v.*v}.$$

(Here, $u.*u$ and $\sqrt{\ }$ represent element-wise vector operations.)

In Figures 3.9a and 3.9b, the functions $\hat{f}$ and $\tilde{f}$, respectively, are drawn as thick solid lines for a representative complementarity problem. Clearly, $\hat{f}$ and $\tilde{f}$ can differ substantially. What is important for solving the complementarity problem, however, is that $\hat{f}$ and $\tilde{f}$ possess the same signs and roots and that $\tilde{f}$ is smoother than $\hat{f}$.

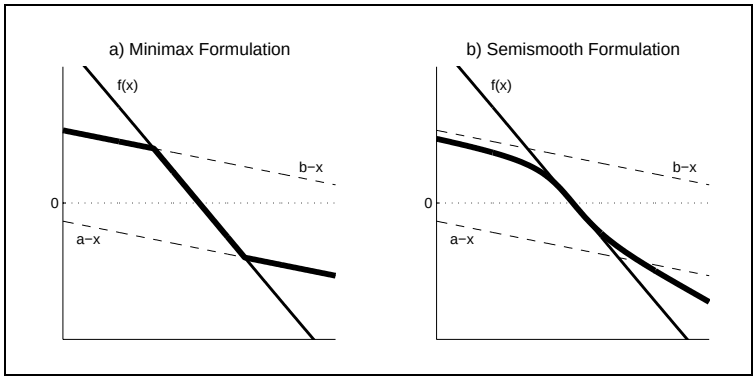

Figure 3.9

The Matlab toolbox accompanying the textbook includes a function `ncpsolve` that solves the complementarity problem by applying Newton's method with safeguarded backstepping to either the minmax or semismooth rootfinding formulations. To apply this function, one defines a Matlab function `f` that returns the function value and Jacobian at arbitrary point, and specifies

the lower and upper bounds, `a` and `b`, and, optionally, a starting value `x`. To solve the complementarity problem using the minmax formulation one writes the Matlab script `x=mcpsolve('f',a,b,x,'minmax')`; to solve the complementarity problem using the semismooth formulation one writes the Matlab script `x=mcpsolve('f',a,b,x,'smooth')`.

In practice, Newton's method applied to either the minmax rootfinding formulation $\hat{f}(x) = 0$ or the semismooth rootfinding formulation $\tilde{f}(x) = 0$ will often successfully solve the complementarity problem $\mathrm{CP}(f, a, b)$. The semismooth formulation is generally more robust than the minmax formulation because it avoids the problematic kinks found in $\tilde{f}$. However, the semismooth formulation also requires more arithmetic operations per iteration.

As an example of a complementarity problem for which the semismooth formulation is successful, but for which the minmax formulation is not, consider the surprisingly difficult complementarity problem $\mathrm{CP}(f, 0, +\infty)$ where

$$f(x) = 1.01 - (x-1)^2.$$

The function f has root at $x = 1 - \sqrt{1.01}$, but this is not a solution to the complementarity problem because it is negative. Also, 0 is not a solution because $f(0) = 0.01$ is positive. The complementarity problem has an unique solution $x = 1 + \sqrt{1.01} \approx 2.005$.

Figure 3.10a displays $\tilde{f}$ (dashed) and $\hat{f}$ (solid) for the complementarity problem and Figure 3.10b magnifies the plot near the origin, making it clear why the problem is hard. Newton's method starting at any value slightly less than 1 will tend to move toward 0. In order to avoid convergence to this false root, Newton's method must take a sufficiently large step to exit the region of attraction. This will not happen with $\tilde{f}$ because 0 poses an upper bound on the positive Newton step. With $\hat{f}$, however, the function is smooth at its local maximum near the origin, meaning that the Newton step can be very large.

To solve the complementarity problem using the semismooth formulation, one codes the function

```
function [fval,fjac] = f(x)
fval = 1.01-(1-x).^2;
fjac = 2*(1-x);
```

and then executes the Matlab script

```
x = mcpsolve('f',0,inf,'smooth');
```

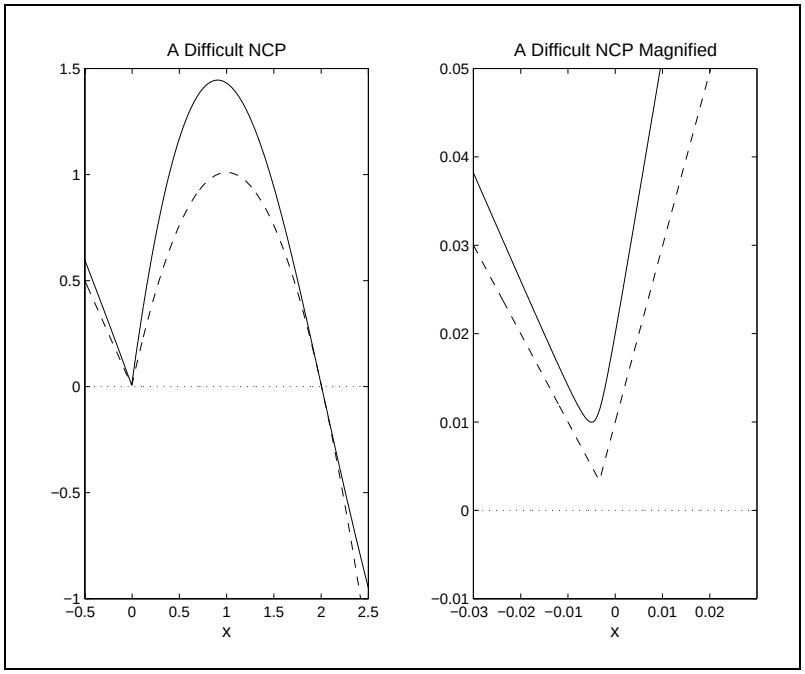

Figure 3.10

To solve the complementarity problem using the minmax formulation, one executes the Matlab script

```
x = mcpsolve('f',0,inf,'minmax');
```

In this example, the semismooth formulation will succcessfully compute the solution of the complementarity problem, but the minmax formulation will not.

Algorithms for solving complementarity problems are still an active area of research, especially for cases that are not well behaved. Algorithms will no doubt continue to improve and existing methods vary considerably in terms of robustness and speed. Our suggestion, however, is to first use a well implemented general purpose root finding algorithm in conjunction with a semismooth formulation. This has the virtue of simplicity and requires only a standard rootfinding utility.

Exercises

1. Consider the function $f : \Re^2 \mapsto \Re^2$ defined by

$$\begin{aligned} f_1(x) &= 200x_1(x_2 - x_1^2) - x_1 + 1 \\ f_2(x) &= 100(x_1^2 - x_2). \end{aligned}$$

Write a Matlab function 'func.m' that takes a column 2-vector `x` as input and returns `f`, a column 2-vector that contains the value of f at `x`, and `d`, a 2 by 2 matrix that contains the Jacobian of f at `x`.

(a) Compute numerically the root of f via Newton's method.

(b) Compute numerically the root of f via Broyden's method.

2. Consider a simple endowment economy with three agents and two goods. Agent i is initially endowed with e_{ij} units of good j and maximizes utility

$$U_i(x) = \sum_{j=1}^{2} a_{ij}(v_{ij} + 1)^{-1} x_{ij}^{v_{ij}+1},$$

subject to the budget constraint

$$\sum_{j=1}^{2} p_j x_{ij} = \sum_{j=1}^{2} p_j e_{ij}.$$

Here, x_{ij} is the amount of good j consumed by agent i, p_j is the market price of good j, and $a_{ij} > 0$ and $v_{ij} < 0$ are preference parameters.

A competitive general equilibrium for the endowment economy is a pair of relative prices, p_1 and p_2, normalized to sum to one, such that all the goods markets clear if each agent maximizes utility subject to his budget constraints.

Compute the competitive general equilibrium for the following parameters:

(i,j)	a_{ij}	v_{ij}	e_{ij}
(1,1)	2.0	-2.0	2.0
(1,2)	1.5	-0.5	3.0
(2,1)	1.5	-1.5	1.0
(2,2)	2.0	-0.5	2.0
(3,1)	1.5	-0.5	4.0
(3,2)	2.0	-1.5	0.0

3. Consider the market for potatoes, which are storable intraseasonaly, but not interseasonaly. In this market, the harvest is entirely consumed over two marketing periods, $i = 1, 2$. Denoting initial supply by s and consumption in period i by c_i, material balance requires that:

   $$s = c_1 + c_2.$$

   Competition among storers possessing perfect foresight eliminate inter-period arbitrage opportunities; thus,

   $$p_1 + \kappa = \delta p_2$$

   where p_i is equilibrium price in period i, $\kappa = 0.2$ is per-period unit cost of storage, and $\delta = 0.9$ is per-period discount factor. Demand, assumed the same across periods, is given by

   $$p_i = c_i^{-5}.$$

   Compute the equilibrium period 1 and period 2 prices for $s = 1$, $s = 2$, and $s = 3$.

4. Provide a formal proof that the complementarity problem $\mathrm{CP}(f, a, b)$ is equivalent to the rootfinding problem $\hat{f}(x) = \min(\max(f(x), a - x), b - x) = 0$ in that both have the same solutions.

5. Commodity X is produced and consumed in three countries. Let quantity q be measured in units and price p be measured in dollars per unit. Demand and supply in the three countries is given by:

	Demand	Supply
Country 1:	$p = 42 - 2q$	$p = 9 + 1q$
Country 2:	$p = 54 - 3q$	$p = 3 + 2q$
Country 3:	$p = 51 - 1q$	$p = 18 + 1q$

The unit costs of transportation are:

	to		
From	Country 1	Country 2	Country 3
Country 1:	0	3	9
Country 2:	3	0	3
Country 3:	6	3	0

   (a) Formulate and solve the linear equation that characterizes competitive equilibrium, assuming that intercountry trade is not permitted.

   (b) Formulate and solve the linear complementarity problem that characterizes competitive spatial equilibrium, assuming that intercountry trade is permitted.

   (c) Using standard measures of surplus, which of the six consumer and producer groups in the three countries gain, and which ones lose, from the introduction of trade.

6. Write a program that solves the following expression for α:

$$\alpha \int_0^\infty \exp(\alpha\lambda - \lambda^2/2) d\lambda = 1$$

and demonstrate that the solution is $\alpha = 0.8399$.

Chapter 4

Finite-Dimensional Optimization

In this chapter we examine methods for optimizing a function with respect to a finite number of variables. In the finite-dimensional optimization problem, one is given a real-valued function f defined on $X \subset \Re^n$ and asked to find an $x^* \in X$ such that $f(x^*) \geq f(x)$ for all $x \in X$. We denote this problem

$$\max_{x \in X} f(x)$$

and call f the objective function, X the feasible set, and x^*, if it exists, a maximum.[1]

Finite-dimensional optimization problems are ubiquitous in Economics. For example, the standard neoclassical models of firm and individual decisionmaking involve the maximization of profit and utility functions, respectively. Competitive static price equilibrium models can often be equivalently characterized as optimization problems in which a hypothetical social planner maximizes total surplus. Finite-dimensional optimization problems arise in econometrics, as in the minimization of the sum of squares or the maximization of a likelihood function. And one also encounters finite-dimensional optimization problems embedded within the Bellman equation that characterizes the solution to continuous-space dynamic optimization models.

There is a close relationship between the finite-dimensional optimization problems discussed in this chapter and the rootfinding and complementarity

[1] We focus our discussion on maximization. To solve a minimization problem, one simply maximizes the negative of the objective function.

problems discussed in the previous chapter. The first-order necessary conditions of an unconstrained problem pose a rootfinding problem; the Karush-Kuhn-Tucker first-order necessary conditions of a constrained optimization problem pose a complementarity problem. The rootfinding and complementarity problems associated with optimization problems are special in that they possess a natural merit function, the objective function itself, which may be used to determine whether iterations are converging on a solution.

Over the years, numerical analysts have studied finite-dimensional optimization problems extensively and have devised a variety of algorithms for solving them quickly and accurately. We begin our discussion with derivative-free methods, which are useful if the objective function is rough or if its derivatives are expensive to compute. We then turn to Newton-type methods for unconstrained optimization, which employ derivatives or derivative estimates to locate an optimum. Univariate unconstrained optimization methods are of particular interest because many multivariate optimization algorithms use the strategy of first determining a linear direction to move in, and then finding the optimal point in that direction. We conclude with a discussion of how to solve constrained optimization problems.

Before proceeding, we review some facts about finite-dimensional optimization and define some terms. By the Theorem of Wierstrass, if f is continuous and X is nonempty, closed, and bounded, then f has a maximum on X. A point $x^* \in X$ is a local maximum of f if there is an ϵ-neighborhood N of x^* such that $f(x^*) \geq f(x)$ for all $x \in N \cap X$. The point x^* is a strict local maximum if, additionally, $f(x^*) > f(x)$ for all $x \neq x^*$ in $N \cap X$. If x^* is a local maximum of f that resides in the interior of X and f is twice differentiable there, then $f'(x^*) = 0$ and $f''(x^*)$ is negative semidefinite. Conversely, if $f'(x^*) = 0$ and $f''(x)$ is negative semidefinite in an ϵ-neighborhood of x^* contained in X, then x^* is a local maximum; if, additionally, $f''(x^*)$ is negative definite, then x^* is a strict local maximum. By the Local-Global Theorem, if f is concave, X is convex, and x^* is a local maximum of f, then x^* is a global maximum of f on X.[2]

[2]These results also hold for minimization, provided one changes concavity of f to convexity and negative (semi) definiteness of f'' to positive (semi) definiteness.

4.1 Derivative-Free Methods

As was the case with univariate rootfinding, optimization algorithms exist that will place progressively smaller brackets around a local maximum of a univariate function. Such methods are relatively slow, but do not require the evaluation of function derivatives and are guaranteed to find a local optimum to a prescribed tolerance in a known number of steps.

The most widely-used derivative-free method is the golden search method. Suppose we wish to find a local maximum of a continuous univariate function $f(x)$ on the interval $[a, b]$. Pick any two numbers in the interior of the interval, say x_1 and x_2 with $x_1 < x_2$. Evaluate the function and replace the original interval with $[a, x_2]$ if $f(x_1) > f(x_2)$ or with $[x_1, b]$ if $f(x_2) \geq f(x_1)$. A local maximum must be contained in the new interval because the endpoints of the new interval are lower than a point on the interval's interior. We can repeat this procedure, producing a sequence of progressively smaller intervals that are guaranteed to contain a local maximum, until the length of the interval is shorter than some desired tolerance level.

A key issue is how to pick the interior evaluation points. Two simple criteria lead to the most widely-used strategy. First, the length of the new interval should be independent of whether the upper or lower bound is replaced. Second, on successive iterations, one should be able to reuse an interior point from the previous iteration so that only one new function evaluation is performed per iteration. These conditions are uniquely satisfied by selecting $x_i = a + \alpha_i(b - a)$, where

$$\alpha_1 = \frac{3 - \sqrt{5}}{2} \text{ and } \alpha_2 = \frac{\sqrt{5} - 1}{2}.$$

The value α_2 is known as the golden ratio, a number dear to the hearts of Greek philosophers and Renaissance artists.

The following Matlab script computes a local maximum of a univariate function f on an interval $[a, b]$ using the golden search method. The script assumes that the user has written a Matlab routine `f` that evaluates the function at an arbitrary point. The script also assumes that the user has specified interval endpoints `a` and `b` and a convergence tolerance `tol`:

```
alpha1 = (3-sqrt(5))/2;
alpha2 = (sqrt(5)-1)/2;
x1 = a+alpha1*(b-a); f1 = f(x1);
x2 = a+alpha2*(b-a); f2 = f(x2);
```

```
d  = alpha1*alpha2*(b-a);
while d>tol
  d = d*alpha2;
  if f2<f1
    x2 = x1; x1 = x1-d;
    f2 = f1; f1 = f(x1);
  else
    x1 = x2; x2 = x2+d;
    f1 = f2; f2 = f(x2);
  end
end
if f2>f1
   x = x2;
else
   x = x1;
end
```

The Matlab toolbox accompanying the textbook includes a function `golden` that computes a local maximum of a univariate function using the golden search method. To apply this function, one defines a Matlab function that returns the value of the optimand at an arbitrary point and specifies the lower and upper bounds for the search interval. For example, to compute a local maximum of $f(x) = x\cos(x^2) - 1$ on the interval $[0, 3]$, one executes the following Matlab script:

```
f = inline('x*cos(x^2)-1');
x = golden(f,0,3)
```

Execution of this script yields the result $x = 0.8083$. As can be seen in Figure 4.1, this point is a local maximum, but not a global maximum in $[0, 3]$. The golden search method is guaranteed to find the global maximum when the function is concave. However, as the present example makes clear, this need not be true when the optimand is not concave.

Another widely-used derivative-free optimization method for multivariate functions is the Nelder-Mead algorithm. The algorithm begins by evaluating the objective function at $n + 1$ points. These $n + 1$ points form a so-called *simplex* in the n-dimensional decision space. This is most easily visualized when x is 2-dimensional, in which case a simplex is a triangle.

At each iteration, the algorithm determines the point on the simplex with the lowest function value and alters that point by reflecting it through the opposite face of the simplex. This is illustrated in Figure 4.2 (Reflection), where the original simplex is lightly shaded and the heavily shaded simplex is

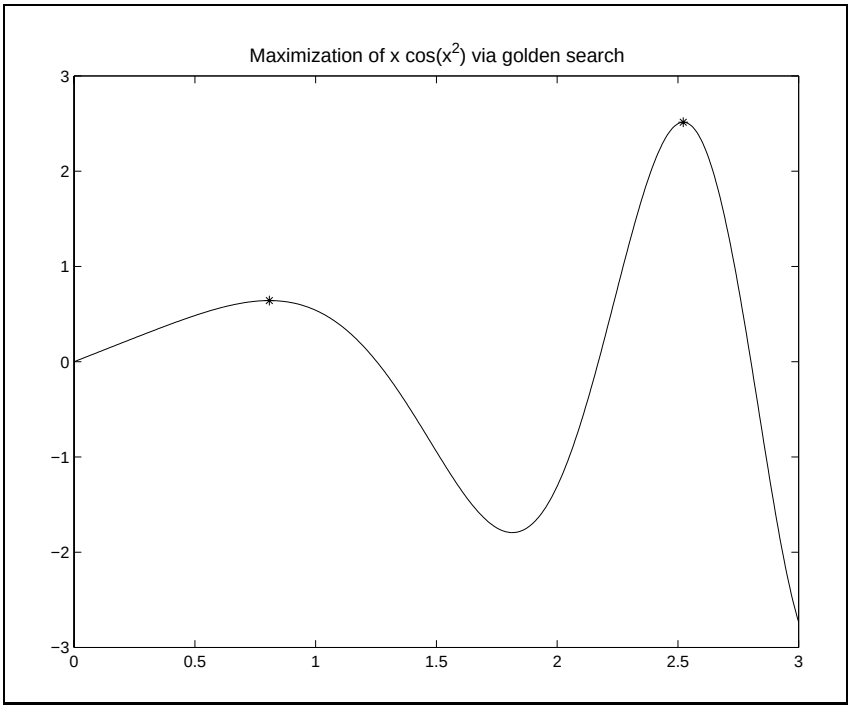

Figure 4.1

the simplex arising from reflecting point A. If the reflection succeeds in finding a new point that is higher than all the others on the simplex, the algorithm checks to see if it is better to expand the simplex further in this direction, as shown in Figure 4.2 (Expansion). On the other hand, if the reflection strategy fails to produce a point that is at least as good as the second worst point, the algorithm contracts the simplex by halving the distance between the original point and its opposite face, as in Figure 4.2 (Contraction). Finally, if this new point is not better than the second worst point, the algorithm shrinks the entire simplex toward the best point, point B in Figure 4.2 (Shrinkage).

One thing that may not be clear from the description of the algorithm is how to compute a reflection. For a point x_i, the reflection is equal to $x_i + 2d_i$ where $x_i + d_i$ is the point in the center of the opposite face of the simplex from x_i. That central point can be found by averaging the n other point of

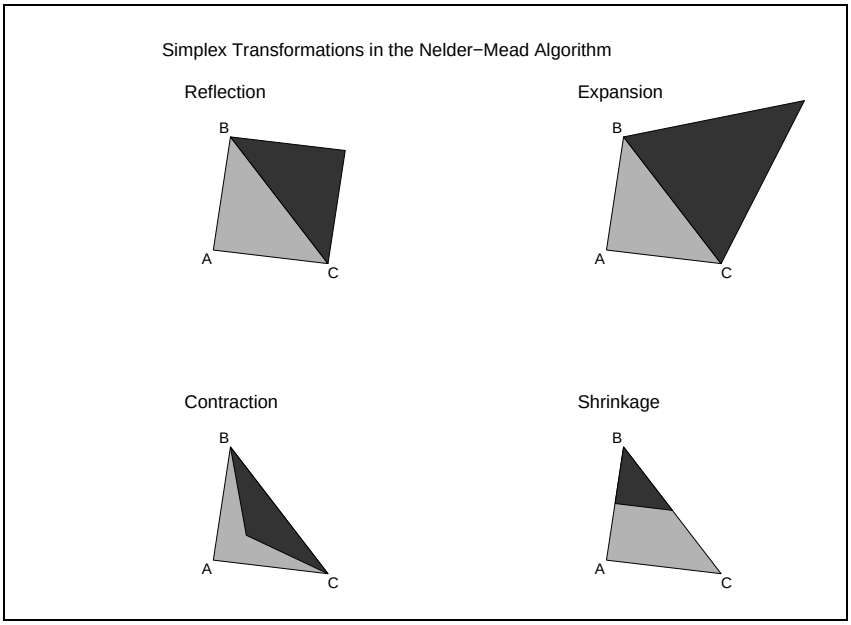

Figure 4.2

the simplex. Denoting the reflection by r_i, this means that

$$r_i = x_i + 2\left(\frac{1}{n}\sum_{j\neq i} x_j - x_i\right) = \frac{2}{n}\sum_{j=1}^{n} x_j - \left(1 + \frac{1}{2}\right) x_i.$$

An expansion can then be computed as

$$1.5r_i - 0.5x_i$$

and a contraction as

$$0.25r_i + 0.75x_i.$$

The Nelder-Mead algorithm is simple, but slow and unreliable. However, if a problem involves only a single optimization or costly function and derivative evaluations, the Nelder-Mead algorithm is worth trying. In many problems an optimization problem that is embedded in a larger problem must be solved repeatedly, with the function parameters perturbed slightly with each iteration. For such problems, which are common is dynamic models,

one generally will want to use a method that moves more quickly and reliably to the optimum, given a good starting point.

The Matlab toolbox accompanying the textbook includes a function `neldmead` that maximizes a multivariate function using the Nelder-Meade method. To apply this function, one defines a Matlab function `f` that returns the function value at an arbitrary point and specifies a starting value `x`. Consider, for example, maximizing the "banana" function $f(x) = -(100x_2) - x_1^2)^2 - (1 - x_1)^2$, so-called because its contours resemble bananas. Assuming a starting value of $(1, 0)$, the Nelder-Meade procedure may be executed in Matlab as follows:

```
f = inline('-100*(x(2)-x(1)^2)^2-(1-x(1))^2');
x = neldmead(f,[1; 0]);
```

Execution of this script yields the result $x = (1, 1)$, which indeed is the global maximum of the function. The contours of the banana function and the path followed by the Nelder-Meade iterates are illustrated in Figure 4.3.

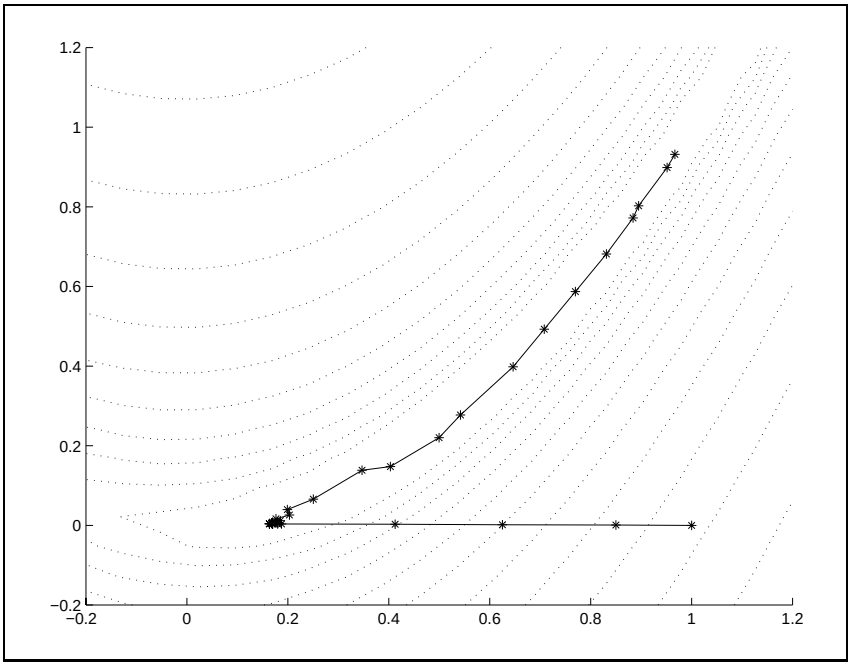

Figure 4.3

4.2 Newton-Raphson Method

The Newton-Raphson method for maximizing an objective function uses successive quadratic approximations to the objective in the hope that the maxima of the approximants will converge to the maximum of the objective. The Newton-Raphson method is intimately related to the Newton method for solving rootfinding problems. Indeed, the Newton-Raphson method is identical to applying Newton's method to compute the root of the gradient of the objective function.

More generally, the Newton-Raphson method begins with the analyst supplying a guess $x^{(0)}$ for the maximum of f. Given $x^{(k)}$, the subsequent iterate $x^{(k+1)}$ is computed by maximizing the second order Taylor approximation to f about $x^{(k)}$:

$$f(x) \approx f(x^{(k)}) + f'(x^{(k)})(x - x^{(k)}) + \tfrac{1}{2}(x - x^{(k)})^\top f''(x^{(k)})(x - x^{(k)}).$$

Solving the first order condition

$$f'(x^{(k)}) + f''(x^{(k)})(x - x^{(k)}) = 0,$$

yields the iteration rule

$$x^{(k+1)} \leftarrow x^{(k)} - [f''(x^{(k)})]^{-1} f'(x^{(k)}).$$

In theory, the Newton-Raphson method converges if f is twice continuously differentiable and if the initial value of x supplied by the analyst is "sufficiently" close to a local maximum of f at which the Hessian f'' is negative definite. There is, however, no generally practical formula for determining what sufficiently close is. Typically, an analyst makes a reasonable guess for the maximum of f and counts his blessings if the iterates converge. The Newton-Raphson method can be robust to the starting value if f is well behaved, for example, if f is globally concave. The Newton-Raphson method, however, can be very sensitive to starting value if the function is not globally concave. Also, in practice, the Hessian f'' must be well-conditioned at the optimum, otherwise rounding errors in the vicinity of the optimum can make it difficult to compute a precise approximate solution.

The Newton-Raphson algorithm has numerous drawbacks. First, the algorithm requires computation of both the first and second derivatives of the objective function. Second, the Newton-Raphson algorithm offers no guarantee that the objective function value may be increased in the direction of

the Newton step. Such a guarantee is available only if the Hessian $f''(x^{(k)})$ is negative definite; otherwise, one may actually move towards a saddle point of f (if the Hessian is indefinite) or even a minimum (if Hessian is positive definite). For this reason, the Newton-Raphson method is rarely used in practice, and then only if the objective function is globally concave.

4.3 Quasi-Newton Methods

Quasi-Newton methods employ a similar strategy to the Newton-Raphson method, but replace the Hessian of the objective (or its inverse) with a negative definite approximation, guaranteeing that function value can be increased in the direction of the Newton step. The most efficient quasi-Newton algorithms employ an approximation to the inverse Hessian, rather than the Hessian itself, in order to avoid performing a linear solve, and employ updating rules that do not require second derivative information to ease the burden of implementation and the cost of computation.

In analogy with the Newton-Raphson method, quasi-Newton methods use a search direction of the form

$$d^{(k)} = -A^{(k)} f'(x^{(k)})$$

where $A^{(k)}$ is an approximation to the inverse Hessian of f at the k^{th} iterate $x^{(k)}$. The vector $d^{(k)}$ is called the Newton or quasi-Newton step.

The more robust quasi-Newton methods do not necessarily take the full Newton step, but rather shorten it or lengthen it in order to obtain improvement in the objective function. This is accomplished by performing a line-search in which one seeks a step length $s > 0$ that maximizes or nearly maximizes $f(x^{(k)} + sd^{(k)})$. Given the computed step length $s^{(k)}$, one updates the iterate as follows:

$$x^{(k+1)} = x^{(k)} + s^{(k)} d^{(k)}.$$

Line search methods are discussed in the following section.

Quasi-Newton method differ in how the Hessian approximation A^k is constructed and updated. The simplest quasi-Newton method sets $A^k = -I$, where I is the identity matrix. This leads to a Newton step that is identical to the gradient of the objective function at the current iterate:

$$d^{(k)} = f'(x^{(k)}).$$

The choice of gradient as a step direction is intuitively appealing because the gradient always points in the direction which, to a first order, promises the greatest increase in f. For this reason, this quasi-Newton method is called the *method of steepest ascent*. The steepest ascent method is simple to implement, but is numerically less efficient in practice than competing quasi-Newton methods that incorporate information regarding the curvature of the objective function.

The most widely-used quasi-Newton methods that employ curvature information produce a sequence of inverse Hessian estimates that satisfy two conditions. First, given that

$$d^{(k)} \approx f''^{-1}(x^{(k)}) \left(f'(x^{(k)} + d^{(k)}) - f'(x^{(k)}) \right),$$

the inverse Hessian estimate A^k is required to satisfy the so-called *quasi-Newton* condition:

$$d^{(k)} = A^{(k)} \left(f'(x^{(k)} + d^{(k)}) - f'(x^{(k)}) \right).$$

Second, the inverse Hessian estimate $A^{(k)}$ is required to be both symmetric and negative-definite, as must be true of the inverse Hessian at a local maximum. The negative definiteness of the Hessian estimate assures that the objective function value can be inreased in the direction of the Newton step.

Two methods that satisfy the quasi-Newton and negative definiteness conditions are the Davidson-Fletcher-Powell (DFP) and Broyden-Fletcher-Goldfarb-Shano (BFGS) updating methods. The DFP method uses the updating scheme

$$A \leftarrow A + \frac{dd^\top}{d^\top u} - \frac{Auu^\top A}{u^\top A u},$$

where

$$d = x^{(k)} - x^{(k-1)}$$

and

$$u = f'(x^{(k)}) - f'(x^{(k-1)}).$$

The BFGS method uses the update scheme

$$A \leftarrow A + \frac{1}{d^\top u} \left(wd^\top + dw^\top - \frac{w^\top u}{d^\top u} dd^\top \right),$$

where $w = d - Au$.

The BFGS algorithm is generally considered superior to DFP, although there are problems for which DFP outperforms BFGS. However, except for the updating formulae, the two methods are identical, so it is easy to implement both and give users the choice.[3]

Quasi-Newton methods are susceptible to certain problems. Notice in the update formulae there is a division by $d^\top u$. If this value becomes very small in absolute value, numerical instabilities will result. It is best to monitor this value and skip updating $A^{(k)}$ if it becomes too small. A useful rule for what is too small is

$$|d^\top u| < \epsilon \, ||d|| \, ||u||,$$

where ϵ is the precision of the computer. An alternative to skipping the update, used in the following implementation, is to reset the inverse Hessian approximant to a scaled negative identity matrix.

The following Matlab script computes the maximum of a user-supplied multivariate function f using the quasi-Newton method. The script assumes that the user has written a Matlab routine `f` that evaluates the function at an arbitrary point and that the user has specified a starting point `x`, an initial guess for the inverse Hessian `A`, a convergence tolerance `tol`, and a limit on the number of iterations `maxit`. The script uses an auxiliary algorithm `optstep` to determine the step length (discussed in the next section). The algorithm also offers the user a choice on how to select the search direction (1-steepest ascent, 2-DFP, 3-BFGS). The algorithm outputs `x`, solution vector, if successful:

```
k = size(x,1);
reset = 0;
[fx0,g0] = f(x);
if abs(g0)<eps, return; end
for it=1:maxit
  d = -A*g0;                          % search direction
  if ((d'*g0)./(d'*d)) < eps0         % must go uphill
    A = -eye(k)./max(abs(fx0),1);     % otherwise use
    d = g0./max(abs(fx0),1);          % steepest ascent
    reset = 1;
```

[3]Modern implementations of quasi-Newton methods store and update the Cholesky factors of the inverse Hessian approximation. This approach is numerically more stable and computationally efficient, but is also more complicated and requires routines to update Cholesky factors.

```
  end
  [s,fx] = optstep(StepMeth,f,x,fx0,g0,d,maxstep,varargin{:});
  if fx<=fx0                          % Step search failure
    if reset
      warning('Iterations stuck in qnewton'), return;
    else                              % Use steepest ascent
      A = -eye(k)./max(abs(fx0),1);
      d = g0./max(abs(fx0),1);
      [s,fx] = optstep(StepMeth,f,x,fx0,g0,d,maxstep,varargin{:});
    end
  end
  d = s*d;
  x = x+d;
  [fx,g] = f(x);
  if ShowIters
    fprintf('qnewton: %4i %16.4f %16.4f %16.4f\n',it,fx,norm(d),norm(g)); end
  % Test convergence
  if all(abs(d)/(abs(x)+eps0)<tol)) | all(abs(g)<eps); return; end
  % Update Inverse Hessian
  u = g-g0; ud = u'*d;
  if SearchMeth==1 | abs(ud)<eps          % Steepest ascent
    A = -eye(k)./max(abs(fx),1);
    reset = 1;
  elseif SearchMeth==2;                   % DFP update
    v = A*u;
    A = A + d*d'./ud - v*v'./(u'*v);
    reset = 0;
  elseif SearchMeth==3;                   % BFGS update
    w = d-A*u; wd = w*d';
    A = A + ((wd + wd') - ((u'*w)*(d*d'))./ud)./ud;
    reset = 0;
  end
  %  Update iteration
  fx0 = fx; g0 = g;
end
```

The Matlab toolbox accompanying the textbook includes a function `qnewton` that maximizes a multivariate function using the quasi-Newton method. To apply this function, one defines a Matlab function `f` that returns the function value at arbitrary point and specifies a starting value `x`. Consider, for example, maximizing the banana function $f(x) = -(100x_2) - x_1^2)^2 - (1 - x_1)^2$ assuming a starting value of $(1, 0)$. To maximize the function using the default DFP Hessian update, one proceeds as follows:

```
f = inline('-100*(x(2)-x(1)^2)^2-(1-x(1))^2');
x = qnewton(f,[1;0]);
```

Execution of this script returns the maximum $x = (1, 1)$ in 18 iterations. To maximize the function using the steepest ascent method, one may override the default update method as follows:

```
optset('qnewton','SearchMeth',1);
x = qnewton(f,[1;0]);
```

Execution of this script fails to find the optimum afer 250 iterations, the default maximum allowable, returning the nonoptimal value $x = (0.82, 0.68)$. The path followed by the quasi-Newton method iterates in these two examples are illustrated in Figure 4.4 and 4.5.

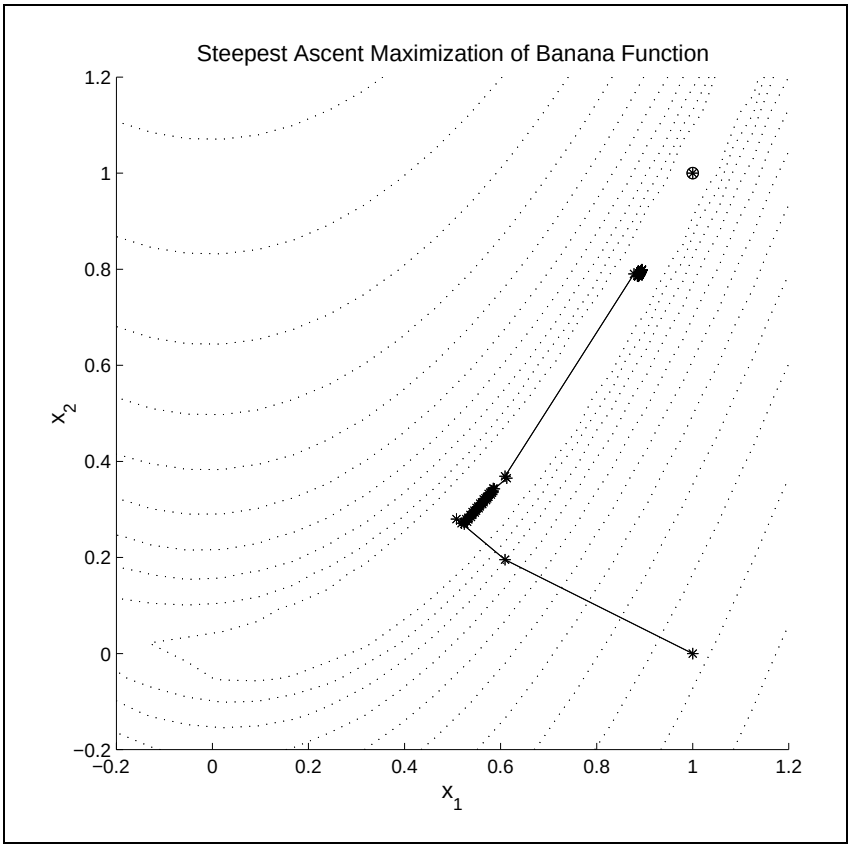

Figure 4.4

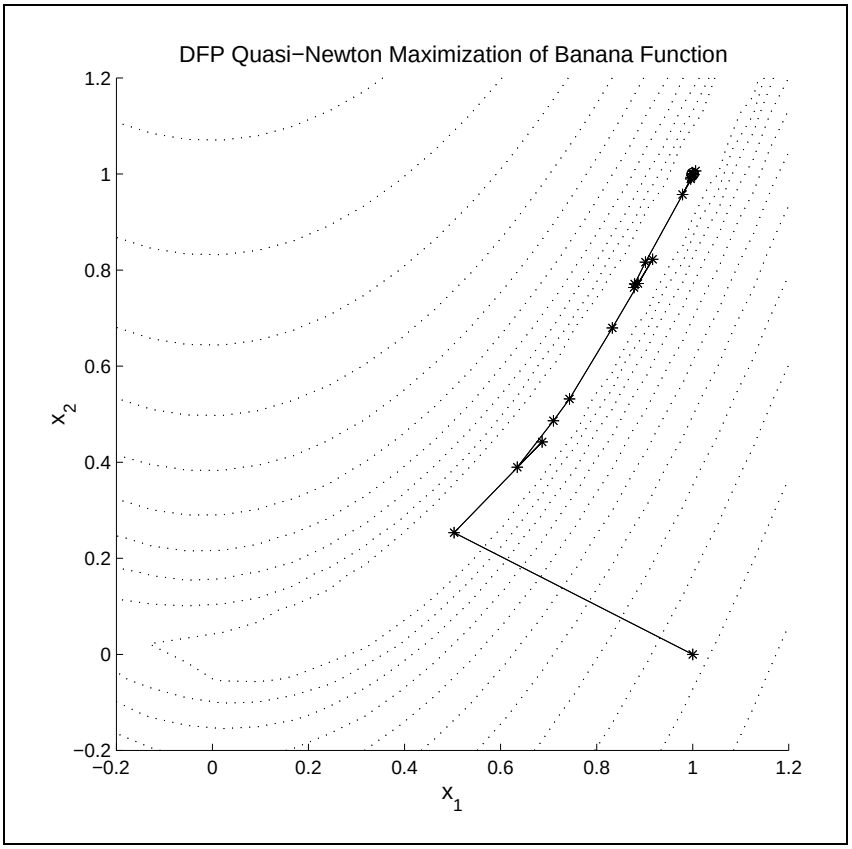

Figure 4.5

4.4 Line Search Methods

Just as was the case with rootfinding problems, it is not always best to take a full Newton step. In fact, it may be better to either stop short or move past the Newton step. If we view the Newton step as defining a *search direction*, performing a one-dimensional search in that direction will generally produce improved results.

In practice, it is not necessary to perform a thorough search for the best point in the Newton direction. Typically, it is sufficient to assure that successive quasi-Newton iterations are raising the value of the objective. A number of different *line search* methods are used in practice, including the golden search method. The golden search algorithm is very reliable, but computa-

tionally inefficient. Two alternative schemes are typically used in practice to perform line searches. The first, known as the Armijo search, is similar to the backstepping algorithm used in rootfinding and complementarity problems. The idea is to find the minimum power j such that

$$\frac{f(x+sd)-f(x)}{s} \geq \mu f'(x)^\top d,$$

where $s = \rho^j$ and $0 < \mu < 0.5$. Note that the left hand side is the slope of the line from the current iteration point to the candidate for the next iteration and the right hand side is the directional derivative at x in the search direction d, that is, the instantaneous slope at the current iteration point. The Armijo approach is to backtrack from a step size of 1 until the slope on the left hand side is a given fraction, μ of the slope on the right hand side.

Another widely-used approach, known as Goldstein search, is to find any value of s that satisfies

$$\mu_0 f'(x)^\top d \leq \frac{f(x+sd)-f(x)}{s} \leq \mu_1 f'(x)^\top d,$$

for some values of $0 < \mu_0 \leq 0.5 \leq \mu_1 < 1$. Unlike the Armijo search, which is both a method for selecting candidate values of the stepsize s and a stopping rule, the Goldstein criteria is simply a stopping rule that can be used with a variety of search approaches.

Figure 4.6 illustrates the typical situation at a given iteration. The figure plots the objective function, expressed as deviations from $f(x)$, i.e., $f(x + sd) - f(x)$, against the step size s in the Newton direction d. The objective function is highlighted and the line tangent to it at the origin has slope equal to the directional derivative $f'(x)^\top d$. The values μ_0 and μ_1 define a cone within which the function value must lie to be considered an acceptable step. In Figure 4.6 the cone is bounded by dashed lines with $\mu_0 = 0.25$ and $\mu_1 = 0.75$. These values are for illustrative purposes and define a far narrower cone than is desirable; typical values are on the order of 0.0001 and 0.9999.

A simple strategy for locating an acceptable point is to first find a point in or above the cone using step doubling (doubling the value of s at each iteration). If a point above the cone is found first, we have a bracket within which points in the cone must lie. We can then narrow the bracket using the golden search method. We call this the `bhhhstep` approach.

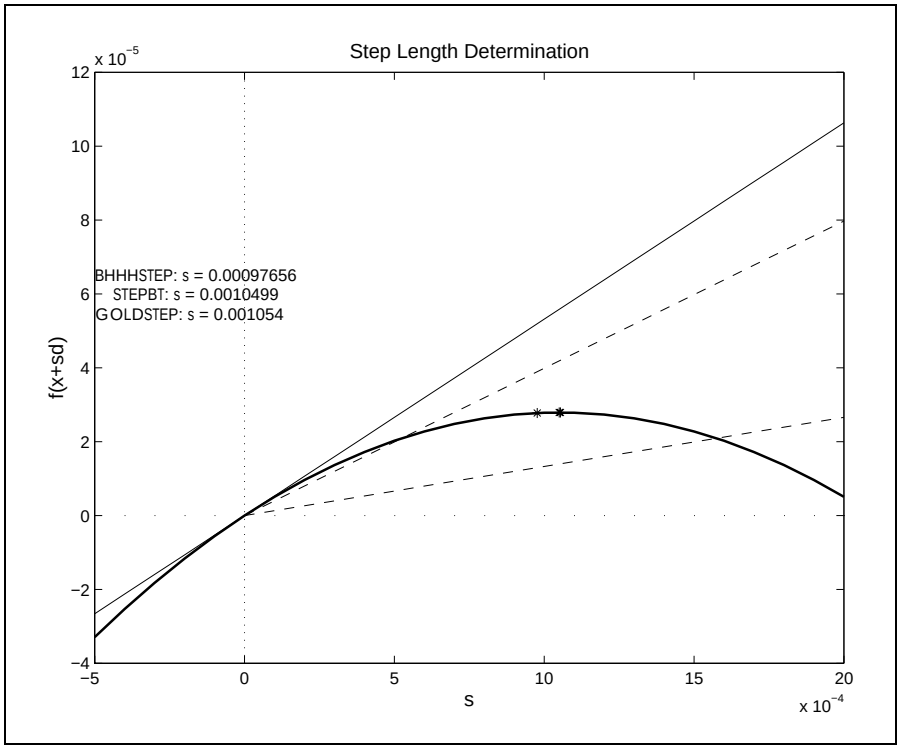

Figure 4.6

Another approach, `stepbt`, checks to see if $s = 1$ is in the cone and, if so, maximizes a quadratic approximation to the objective function in the Newton direction constructed from knowledge of $f(x^{(k)})$, $f'(x^{(k)})d^{(k)}$, and $f(x^{(k)} + d^{(k)})$. If the computed step $s^{(k)}$ is acceptable, it is taken. Otherwise, the algorithm maximizes a cubic approximation to the objective function in the Newton direction constructed from knowledge of the $f(x^{(k)} + s^{(k)}d^{(k)})$ and the three pieces of information used to the construct the quadratic approximant. `stepbt` is fast and generally gives good results. It is recommended as the default lines search procedure for general maximization algorithms.

In Figure 4.6 we have included three stars representing the step lengths determined by `stepbhhh`, `stepbt` and our implementation of the golden search step length maximizer, `stepgold` (also listed below). `stepgold` first brackets a maximum in the direction d and then uses the golden search approach to narrow the bracket. This method differs from the other two in that it termi-

nates when the size of the bracket is less than a specified tolerance (here set at 0.0004).

In this example, the three methods took 11, 4 and 20 iterations to find an acceptable step length, respectively. Notice that `stepbt` found the maximum in far fewer steps than did `stepgold`. This will generally be true when the function is reasonably smooth and hence well approximated by a cubic function. It is difficult to make generalizations about the performance of the step line search algorithm, however. In this example, the step size was very small, so both `stepbhhh` and `stepgold` take many iterations to get the order of magnitude correct. In many case, if the initial distance is well chosen, the step size will typically be close to unity in magnitude, especially as the maximizer approaches the optimal point. When this is true, the advantage of `stepbt` is less important. Having said all of that, we recommend `stepbt` as a default. We have also implemented our algorithm to use `stepgold` if the other methods fail.

4.5 Special Cases

Two special cases arise often enough in economic practice (especially in econometrics) to warrant additional discussion. The non-linear least squares and the maximum likelihood problems have objective functions with special structures that give rise to their own special quasi-Newton methods. The special methods differ from other Newton and quasi-Newton methods only in the choice of the matrix used to approximate the Hessian. Because these problems generally arise in the context of statistical applications, we alter our notation to conform with the conventions for those applications. The optimization takes place with respect to a k-dimensional parameter vector θ and n will refer to the number of observations.

The nonlinear least squares problem takes the form

$$\min_{\theta} \tfrac{1}{2} f(\theta)^\top f(\theta)$$

where $f : \Re^k \to \Re^n$ (the $\frac{1}{2}$ is for notational convenience). The gradient of this objective function is

$$\sum_{i=1}^{n} f_i'(\theta)^\top f_i(\theta) = f'(\theta)^\top f(\theta).$$

The Hessian of the objective function is

$$f'(\theta)^\top f'(\theta) + \sum_{i=1}^{n} f_i(\theta) \frac{\partial^2 f(\theta)}{\partial\theta\partial\theta^\top}.$$

If we ignore the second term in the Hessian, we are assured of having a positive definite matrix with which to determine the search direction:

$$d = -[f'(\theta)^\top f'(\theta)]^{-1} f'(\theta)^\top f(\theta).$$

All other aspects of the problem are identical to the quasi-Newton methods already discussed, except for the adjustment to minimization. It is also worth pointing out that, in typical applications, $f(\theta)$ is an error terms with expectation 0. Assuming that the usual central limit assumptions apply to the error term, the inverse of the approximate Hessian

$$[f'(\theta)^\top f'(\theta)]^{-1},$$

can be used as a covariance estimator for θ.

Maximum likelihood problems are specified by a choice of a distribution function for the data, y, that depends on a parameter vector, θ. The log-likelihood function is the sum of the logs of the likelihoods of each of the data points:

$$l(\theta; y) = \sum_{i=1}^{n} \ln f(\theta; y_i).$$

The score function is defined as the matrix of derivatives of the log-likelihood function evaluated at each observation:

$$s_i(\theta; y) = \frac{\partial l(\theta; y_i)}{\partial\theta}.$$

(viewed as a matrix, the score function is $n \times k$).

A well-known result in statistical theory is that the expectation of the inner product of the score function is equal to the negative of the expectation of the second derivative of the likelihood function, which is known as the information matrix. Either the information matrix or the sample average of the inner product of the score function provides a positive definite matrix that can be used to determine a search direction. In the later case the search direction is defined by

$$d = -[s(\theta; y)^\top s(\theta, y)]^{-1} s(\theta, y)^\top 1_n.$$

This approach is known as the *modified method of scoring*. As in the case of the nonlinear least squares, a covariance estimator for θ is immediately available using

$$[s(\theta; y)^\top s(\theta, y)]^{-1}.$$

4.6 Constrained Optimization

The simplest constrained optimization problem involves the maximization of an objective function subject to simple bounds on the choice variable:

$$\max_{a \leq x \leq b} f(x).$$

According to the Karush-Kuhn-Tucker theorem, if f is differentiable on $[a, b]$, then x^* is a constrained maximum for f only if it solves the complementarity problem $\mathrm{CP}(f', a, b)$:

$$\begin{aligned} &a_i \leq x_i \leq b_i \\ &x_i > a_i \;\Rightarrow\; f_i'(x) \geq 0 \\ &x_i < b_i \;\Rightarrow\; f_i'(x) \leq 0. \end{aligned}$$

Conversely, if f is concave and differentiable on $[a, b]$ and x^* solves the complementarity problem $\mathrm{CP}(f'a, b)$, then x^* is a constrained maximum of f; if additionally f is strictly concave on $[a, b]$, then the maximum is unique.

Two bounded maximization problems are displayed in Figure 4.7. In this figure, the bounds are displayed with dashed lines and the objective function with a solid line. In Figure 4.7A the objective function is concave and achieves its unique global maximum on the interior of the feasible region. At the maximum, the derivative of f must be zero, for otherwise one could improve the objective by moving either up or down, depending on whether the derivative is positive or negative. In Figure 4.7B we display a more complicated case. Here, the objective function is convex. It achieves a global maximum at the lower bound and a local, non-global maximum at the upper bound. It also achieves a global minimum in the interior of the interval.

In Figure 4.8 we illustrate the complementarity problem presented by the Karush-Kuhn-Tucker conditions associated with the bounded optimization problems in Figure 4.7. The complementarity problems are represented in their equivalent rootfinding formulation $\min(\max(f'(x), a - x), b - x) = 0$. In Figure 4.8A we see that the Karush-Kuhn-Tucker conditions possess an

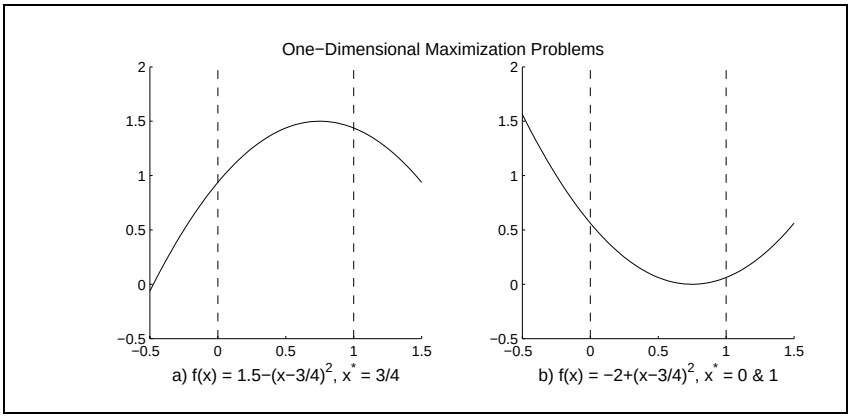

Figure 4.7

unique solution at the unique global maximum of f. In Figure 4.8B there are three solutions to the Karush-Kuhn-Tucker conditions, corresponding to the two local maxima and the one local minimum of f on $[a, b]$. These figures illustrate that one may reliably solve a bounded maximization problem using standard complementarity methods only if the objective function is concave. Otherwise, the complementary algorithm could lead to local, non-global maxima or even minima.

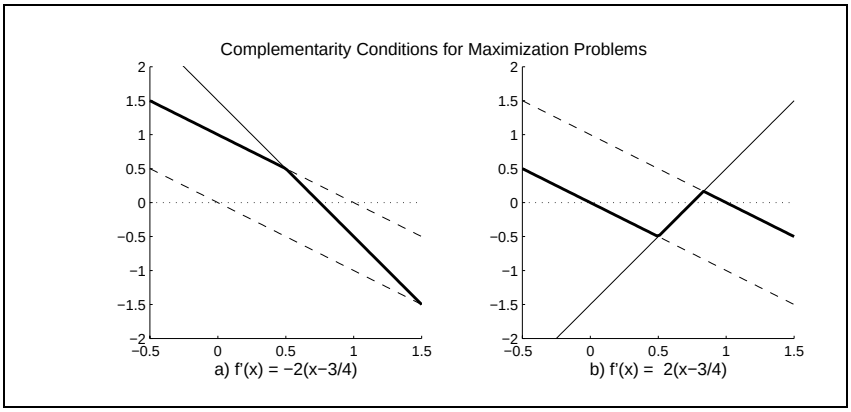

Figure 4.8

The sensitivity of the optimal value of the objective function f^* to changes in the bounds of the bounded optimization problem are relatively easy to characterize. According to the Envelope theorem,

$$\begin{aligned}\frac{df^*}{da} &= \min\left(0, f'(x^*)\right)\\ \frac{df^*}{db} &= \max\left(0, f'(x^*)\right).\end{aligned}$$

More generally, if f, a, and b all depend on some parameter p, then

$$\frac{df^*}{dp} = \frac{\partial f}{\partial p} + \min\left(0, \frac{\partial f}{\partial x}\right)\frac{da}{dp} + \max\left(0, \frac{\partial f}{\partial x}\right)\frac{db}{dp},$$

where the derivatives of f, a, and b are evaluated at (x^*, p).

The most general constrained finite-dimensional optimization problem that we consider is

$$\min_{a\leq x\leq b} f(x), \text{ s.t. } R(x) \begin{matrix}\leqq\\ \geqq\end{matrix} r,$$

where $R : [a, b] \to \Re^m$.

According to the Karush-Kuhn-Tucker Theorem, a regular point x maximizes f subject to the general constraints only if there is a vector $\lambda \in \Re^n$ such that (x, λ) solves the complementarity problem

$$CP\left(\begin{bmatrix} f'(x) - R'(x)\lambda^\top \\ R(x) - r \end{bmatrix}, \begin{bmatrix} a \\ p \end{bmatrix}, \begin{bmatrix} b \\ q \end{bmatrix}\right)$$

where

$$p_i = \begin{cases} 0 & \text{if } i \text{ is a ``}\leq\text{'' constraint} \\ -\infty & \text{otherwise} \end{cases}$$

and

$$q_i = \begin{cases} 0 & \text{if } i \text{ is a ``}\geq\text{'' constraint} \\ \infty & \text{otherwise} \end{cases}.$$

A point x is regular if the gradients of all constraint functions R_i that satisfy $R_i(x) = r_i$ are linearly independent.[4] Conversely, if f is concave, R is convex

[4]The regularity conditions may be omitted if either the constraint function R is linear, or if f is concave, R is convex, and the feasible set has nonempty interior.

and (x, λ) satisfies the Karush-Kuhn-Tucker conditions, then x solves the general constrained optimization problem.

In the Karush-Kuhn-Tucker conditions, the λ_i are called Lagrangian multipliers or shadow prices. The significance of the shadow prices is given by the Envelope Theorem, which asserts that under mild regularity conditions,

$$\frac{\partial f^*}{\partial r} = \lambda,$$

that is, λ_i is the rate at which the optimal value of the objective will change with changes in the constraint constant r_i. The sensitivity of the optimal value of the objective function f^* to changes in the bounds on the choice variable are given by:

$$\begin{aligned} \frac{df^*}{da} &= \min\left(0, f'(x) - R'(x)\lambda^\top\right) \\ \frac{df^*}{db} &= \max\left(0, f'(x) - R'(x)\lambda^\top\right). \end{aligned}$$

The Karush-Kuhn-Tucker complementarity conditions typically have a natural arbitrage interpretation. Consider the problem of maximizing profits from certain economic activities when the activities employ fixed factors or resources that are available in limited supply. Specifically, suppose $x_1, x_2, \ldots, x_n$ are the levels of n economic activities, which must be nonnegative, and the objective is to maximize profit $f(x)$ generated by those activities. Also suppose that these activities employ m resources and that the usage of the i^{th} resource $R_i(x)$ cannot exceed a given availability r_i. Then λ_i^* represents the opportunity cost or shadow price of the i^{th} resource and

$$MP_j = \frac{\partial f}{\partial x_j} - \sum_i \lambda_i^* \frac{\partial R_i}{\partial x_j}$$

represents the economic marginal profit of the j^{th} activity, accounting for the opportunity cost of the resources employed in the activity. The Karush-Kuhn-Tucker conditions may thus be interpreted as follows:

$x_j \geq 0$	activity levels are nonnegative
$MP_j \leq 0$	otherwise, raise profit by raising x_j
$x_j > 0 \Rightarrow MP_j \geq 0$	otherwise, raise profit by lowering x_j
$\lambda_i^* \geq 0$	Shadow price of resource is nonnegative
$R_i(x) \leq r_i$	resource use cannot exceed availability
$\lambda_i > 0 \Rightarrow R_i(x) = r_i$	valuable resources should not be wasted

There are many approaches to solving general optimization problems that would take us beyond what we can hope to accomplish in this book. Solving general optimization problems is difficult and the best advice we can give here is that you should obtain a good package and use it. However, if your problem is reasonably well behaved in the sense that the Karush-Kuhn-Tucker are both necessary and sufficient, then the problem is simply to solve the Karush-Kuhn-Tucker conditions. This means writing the Karush-Kuhn-Tucker conditions as a complementarity problem and solving the problem using the methods of the previous chapter.

Exercises

1. Consider the *Quadratic Programming* problem

$$\begin{aligned} \max_x \quad & \tfrac{1}{2}x'Dx + c'x \\ s.t. \quad & Ax \leq b \\ & x \geq 0 \end{aligned}$$

where D is a symmetric $n \times n$ matrix, A is an $m \times n$ matrix, b is an m-vector.

   (a) Write the Karush-Kuhn-Tucker necessary conditions as a linear complementarity problem.

   (b) What condition on D will guarantee that the Karush-Kuhn-Tucker conditions are sufficient for optimality?

2. A consumer's preferences over the commodities x_1, x_2, and x_3 are characterized by the Stone-Geary utility function

$$U(x) = \sum_{i=1}^{3} \beta_i \ln(x_i - \gamma_i)$$

where $\beta_i > 0$ and $x_i > \gamma_i \geq 0$. The consumer wants to maximize his utility subject to the budget constraint

$$\sum_{i=1}^{3} p_i x_i \leq I$$

where $p_i > 0$ denotes the price of x_i, I denotes income, and $I - \sum_{i=1}^{3} p_i \gamma_i > 0$.

(a) Write the Karush-Kuhn-Tucker necessary conditions for the problem.

(b) Verify that the Karush-Kuhn-Tucker conditions are sufficient for optimality.

(c) Derive analytically the associated demand functions.

(d) Derive analytically the shadow price and interpret its meaning.

(e) Prove that the consumer will utilize his entire income.

3. Derive and interpret the Karush-Kuhn-Tucker conditions for the classical transportation problem:

$$\begin{aligned} \min \quad & \sum_{\imath=1}^{n} \sum_{\jmath=1}^{m} x_{\imath\jmath} x_{\imath\jmath} \\ s.t. \quad & \sum_{\imath=1}^{n} x_{\imath\jmath} \geq d_{\jmath} \quad \jmath = 1, \ldots, m \\ & \sum_{\jmath=1}^{m} x_{\imath\jmath} \leq s_{\imath} \quad \imath = 1, \ldots, n \\ & x_{\imath\jmath} \geq 0 \qquad \imath = 1, \ldots, n, \jmath = 1, \ldots, m \end{aligned}$$

State sufficient conditions for the transportation problem to have an optimal feasible solution.

4. Demand for a commodity in regions A and B is given by:

$$\begin{aligned} &\textit{Region } A: \quad p = 200 - 2q \\ &\textit{Region } B: \quad p = 100 - 4q \end{aligned}$$

Supply is given by:

$$\begin{aligned} &\textit{Region } A: \quad p = 20 + 8q \\ &\textit{Region } B: \quad p = 10 + 6q. \end{aligned}$$

The transportation cost between regions is \$10 per unit.

Formulate an optimization problem that characterizes the competitive spatial price equilibrium. Derive, but do not solve, the Karush-Kuhn-Tucker conditions. Interpret the shadow prices.

5. Consider a vector of n random assets with expected return $\mu_{n\times 1}$ and variance $\Sigma_{n\times n}$. Formulate a quadratic program whose solution is the Markowitz E-V efficient portfolio $x_{n\times 1}$ whose expected return is at least r^*. Derive the Karush-Kuhn-Tucker conditions for the program. Interpret the Lagrangian multiplier and explain its relation to the risk aversion parameter ϕ of the objective function $\mu'\cdot x - \phi\cdot x'\Sigma x$ of Freund's portfolio choice model.

6. Consider the nonlinear programming problem

$$\begin{aligned} max_{x_1,x_2} \quad & x_2^2 - 2x_1 - x_1^2 \\ s.t. \quad & x_1^2 + x_2^2 \leq 1 \\ & x_1 \geq 0, x_2 \geq 0. \end{aligned}$$

   (a) Write the Karush-Kuhn-Tucker necessary conditions for the problem.

   (b) What points satisfy the Karush-Kuhn-Tucker necessary conditions.

   (c) Are the Karush-Kuhn-Tucker conditions sufficient for optimality?

   (d) How do you know that problem possesses an optimum?

   (e) Determine the optimum, if any.

7. A tomato processor operates two plants whose hourly variable costs (in dollars) are, respectively,

$$\begin{aligned} c_1 &= 80 + 2.0x_1 + 0.001x_1^2 \\ c_2 &= 90 + 1.5x_2 + 0.002x_2^2, \end{aligned}$$

   where x_i is the number of cases produced per hour at plant i. In order to meet contractual obligations, he must produce at a rate of at least 2000 cases per hour ($x_1 + x_2 \geq 2000$.) He wishes to do so at minimal cost.

(a) Write the Karush-Kuhn-Tucker necessary conditions for the problem.

(b) Verify that the Karush-Kuhn-Tucker conditions are sufficient for optimality.

(c) Determine the optimal levels of production.

(d) Determine the optimal value of the shadow price and interpret its meaning.

8. Consider the problem of allocating a scarce resource, the total supply of which is $b > 0$, among n tasks with separable rewards:

$$\begin{aligned} \max_{x_1,x_2,\ldots,x_n} \quad & f_1(x_1) + f_2(x_2) + \ldots + f_n(x_n) \\ s.t. \quad & x_1 + x_2 + \ldots + x_n \leq b \\ & x_1 \geq 0, x_2 \geq 0, \ldots, x_n \geq 0. \end{aligned}$$

Assume each f_i is strictly increasing and differentiable but not necessarily concave.

(a) How do you know that problem possesses an optimum?

(b) Write the Karush-Kuhn-Tucker necessary conditions.

(c) Prove that the scarce resource will be completely utilized.

(d) Interpret the shadow price associated with the resource constraint.

(e) Given a marginal increase in the supply of the resource, to which task(s) would you allocate the additional amount.

9. Consider a one-output two-input production function

$$y = f(x_1, x_2) = x_1^2 + x_2^2.$$

Given the prices of inputs 1 and 2, w_1 and w_2, the minimum cost of producing a given level of output, $\bar{y}$, is obtained by solving the constrained optimization problem

$$\begin{aligned} \min_{x_1,x_2} \quad & C = w_1 \cdot x_1 + w_2 \cdot x_2 \\ s.t. \quad & f(x_1, x_2) \geq \bar{y}. \end{aligned}$$

Letting λ denote the shadow price associated with the production constraint, answer the following questions:

(a) Write the Karush-Kuhn-Tucker necessary conditions.

(b) Find explicit expressions for the optimal x_1^*, x_2^*, and C^*.

(c) Find an explicit expression for the optimal λ^* and interpret its meaning.

(d) Differentiate the expression for C^* to confirm that $\frac{\partial C^*}{\partial \bar{y}} = \lambda^*$.

10. A salmon cannery produces Q 1-lb. cans of salmon according to a technology given by $Q = 18K^{\frac{1}{4}}L^{\frac{1}{3}}$, where capital K is fixed at 16 units in the shortrun and labor L may be hired in any quantity at a wage rate of w dollars per unit. Each unit of output provides a profit contribution of 1 dollar.

(a) Derive the firm's shortrun demand for labor.

(b) If $w = 3$, how much would the firm be willing to pay to rent a unit of capital.

11. Consider the nonlinear programming problem

$$\begin{aligned} \min_{x_1,\ldots,x_4} \quad & x_1^{0.25} x_3^{0.50} x_4^{0.25} \\ s.t. \quad & x_1 + x_2 + x_3 + x_4 \geq 4 \\ & x_1, x_2, x_3, x_4 \geq 0. \end{aligned}$$

(a) What can you say about the optimality of the point $(1, 0, 2, 1)$?

(b) Does this program possess all the correct curvature properties for the Karush-Kuhn-Tucker conditions to be sufficient for optimality throughout the feasible region? Why or why not?

(c) How do you know that problem possesses an optimal feasible solution?

12. Consider the non-linear programming problem

$$\begin{aligned} \min_{x_1,x_2} \quad & 2x_1^2 - 12x_1 + 3x_2^2 - 18x_2 + 45 \\ s.t. \quad & 3x_1 + x_2 \leq 12 \\ & x_1 + x_2 \leq 6 \\ & x_1, x_2 \geq 0. \end{aligned}$$

The optimal solution to this problem is: $x_1^* = 3$ and $x_2^* = 3$.

(a) Verify that the Karush-Kuhn-Tucker conditions are satisfied by this solution.

(b) Determine the optimal values for the shadow prices λ_1 and λ_2 associated with the structural constraints, and interpret λ_1^* and λ_2^*.

(c) If the second constraint were changed to $x_1 + x_2 \leq 5$, what would be the effect on the optimal values of x_1, x_2, λ_1, and λ_2?

13. Suppose that the probability density function of a non-negative random variable, y, is

$$\exp(-y_i/\mu_i)/\mu_i$$

where $\mu_i = X_i\beta$ for some observable data X_i.

(a) Show that the first order conditions for the maximum likelihood estimator of β can be written as

$$\sum \frac{{X_i}^\top X_i}{(X_i\beta)^2}\beta = \sum \frac{{X_i}^\top y_i}{(X_i\beta)^2}.$$

(b) Use this result to define a recursive algorithm to estimate β.

(c) Write a Matlab function of the form `[beta,sigma]=example(y,X)` that computes the maximum likelihood estimator of β and its asymptotic covariance matrix Σ. The function should be a stand-alone procedure (i.e., do not call any optimization or root-finding solvers) that implements the recursive algorithm.

(d) Show that the recursive algorithm can be interpreted as a quasi-Newton method. Explain fully.

14. Write a Matlab function that is passed a vector of observations (of positive numbers) and returns the maximum likelihood estimates of θ and their covariance matrix for the two-parameter gamma function:

$$f(x;\theta) = \frac{\theta_2^{\theta_1} x^{\theta_1 - 1} e^{-\theta_2 x}}{\Gamma(\theta_1)}.$$

Hint: Formulate the problem as a maximization of the log-likelihood. Note that the first and second derivatives of the log of the Γ function are the psi and trigamma functions. The Matlab toolbox contains procedures to evaluate these special functions.

15. Continuing in the vein of the last problem, reformulate the likelihood function of the two-parameter Gamma distribution in terms of θ_1 and $\psi = \theta_1/\theta_2$.

    (a) Solve explicitly for the optimal ψ, and express the likelihood function in terms of θ_1 and the data alone.

    (b) Write a Matlab function that maximizes the resulting univariate likelihood function using algorithm `golden` provided in the toolbox.

    (c) Write a Matlab function that maximizes the resulting univariate likelihood function using algorithm `newton` provided in the toolbox.

    (d) The maximum likelihood estimator of θ depends on the data only through $Y_1 = \frac{1}{n}\sum_{i=1}^{n} x_i$, the arithmetic mean, and $Y_2 = \exp(\frac{1}{n}\sum_{i=1}^{n} \ln(x_i))$, the geometric mean ($Y_1$ and Y_2 are known as sufficient statistics for θ). Plot θ_1 as a function of $\ln(Y_1/Y_2)$.

Chapter 5

Numerical Integration and Differentiation

In many computational economic applications, one must compute the definite integral of a real-valued function f with respect to a "weighting" function w over an interval I of $\Re^n$:

$$\int_I f(x)w(x)\,dx.$$

The weighting function may be the identity, $w \equiv 1$, in which case the integral represents the area under the function f. In other applications, w may be the probability density of a random variable $\tilde{X}$, in which case the integral represents the expectation of $f(\tilde{X})$.

In this chapter, we discuss three classes of numerical integration or *numerical quadrature* methods. All methods approximate the integral with a weighted sum of function values:

$$\int_I f(x)w(x)\,dx \approx \sum_{i=0}^{n} w_i f(x_i)$$

The methods differ only in how the *quadrature weights* w_i and the *quadrature nodes* x_i are chosen. Newton-Cotes methods approximate the integrand f between nodes using low order polynomials, and sum the integrals of the polynomials to estimate the integral of f. Newton-Cotes methods are easy to implement, but are not particularly efficient for computing the integral of a smooth function. Gaussian quadrature methods choose the nodes and weights to satisfy moment matching conditions, and are more powerful than

Newton-Cotes methods if the integrand is smooth. Monte Carlo and quasi-Monte Carlo integration methods use 'random' or 'equidistributed' nodes, and are simple to implement and very useful if the integration domain is of high dimension.

In this chapter, we also present an overview of how to compute *finite difference* approximations for the derivatives of a real-valued function. As we have seen in previous chapters, it is often desirable to compute derivatives numerically because analytic derivative expressions are difficult or impossible to derive, or expensive to evaluate. Finite difference methods can also be used to solve differential equations, which arise frequently in dynamic economic models, especially models formulated in continuous time. In this chapter, we introduce numerical methods for differential equations and illustrate their application to *initial value problems*.

5.1 Newton-Cotes Methods

Newton-Cotes quadrature methods are designed to approximate the integral of a real-valued function f defined on a bounded interval $[a, b]$ of the real line. Newton-Cotes methods approximate the integrand f between nodes using low order polynomials, and sum the integrals of the polynomials to form an estimate the integral of f. Two Newton-Cotes rules are widely used in practice: the trapezoid rule and Simpson's rule. Both rules are very easy to implement and are typically adequate for computing the area under a continuous function.

The simplest way to compute an approximate integral of a real-valued function f over a bounded interval $[a, b] \subset \Re$ is to partition the interval into subintervals of equal length, approximate f over each subinterval using a straight line segment that linearly interpolates the function values at the subinterval endpoints, and then sum the areas under the line segments. This is the so-called trapezoid rule, which draws its name from the fact that the area under f is approximated by a series of trapezoids.

More formally, let $x_i = a + ih$ for $i = 0, 1, 2, \ldots, n$, where $h = (b - a)/n$. The nodes x_i divide the interval $[a, b]$ into n subintervals of equal length h. Over the i^{th} subinterval, $[x_{i-1}, x_i]$, the function f may be approximated by the line segment passing through the two graph points $(x_{i-1}, f(x_{i-1}))$ and $(x_i, f(x_i))$. The area under this line segment defines a trapezoid that provides

an estimate of the area under f over this subinterval:

$$\int_{x_{i-1}}^{x_i} f(x)\ dx \approx \int_{x_{i-1}}^{x_i} \hat{f}(x)\ dx = \frac{h}{2}[f(x_{i-1}) + f(x_i)].$$

Summing up the areas of the trapezoids across subintervals yields the trapezoid rule:

$$\int_a^b f(x)\ dx \approx \sum_{i=0}^{n} w_i\ f(x_i)$$

where $w_0 = w_n = h/2$ and $w_i = h$, otherwise.

The trapezoid rule is simple and robust. Other Newton-Cotes methods will be more accurate if the integrand f is smooth. However, the trapezoid rule will often be more accurate if the integrand exhibits discontinuities in its first derivative, which can occur in economic applications exhibiting corner solutions. The trapezoid rule is said to be first order exact because in theory it exactly computes the integral of any first order polynomial, that is, a line. In general, if the integrand is smooth, the trapezoid rule yields an approximation error that is $O(1/n^2)$, that is, the error shrinks quadratically with the number of subintervals.

Simpson's rule is based on piece-wise quadratic, rather than piece-wise linear, approximations to the integrand f. More formally, let $x_i = a + ih$ for $i = 0, 1, 2, \ldots, n$, where $h = (b-a)/n$ and n is even. The nodes x_i divide the interval $[a, b]$ into an even number n of subintervals of equal length h. Over the j^{th} pair of subintervals, $[x_{2j-2}, x_{2j-1}]$ and $[x_{2j-1}, x_{2j}]$, the function f may be approximated by the unique quadratic function $\hat{f}_j$ that passes through the three graph points $(x_{2j-2}, f(x_{2j-2}))$ $(x_{2j-1}, f(x_{2j-1}))$, and $(x_{2j}, f(x_{2j}))$. The area under this quadratic function provides an estimate of the area under f over the subinterval:

$$\int_{x_{2j-2}}^{x_{2j}} f(x)\ dx \approx \int_{x_{2j-2}}^{x_{2j}} \hat{f}_j(x)\ dx = \frac{h}{3}\left(f(x_{2j-2}) + 4f(x_{2j-1}) + f(x_{2j})\right).$$

Summing up the areas under the quadratic approximants across subintervals yields Simpson's rule:

$$\int_a^b f(x)\ dx \approx \sum_{i=0}^{n} w_i f(x_i)$$

where $w_0 = w_n = h/3$ and, otherwise, $w_i = 2h/3$ if i is even and $w_i = 4h/3$ if i is odd.

Simpson's rule is almost as simple as the trapezoid rule, and thus not much harder to program. Simpson's rule, moreover, will yield more accurate approximations if the integrand is smooth. Even though Simpson's rule is based on locally quadratic approximation of the integrand, it is third order exact. That is, it exactly computes the integral of any third order (e.g., cubic) polynomial. In general, if the integrand is smooth, Simpson's rule yields an approximation error that is $O(1/n^4)$, and thus falls at twice the geometric rate as the error associated with the trapezoid rule. Simpson's rule is the Newton-Cotes rule most often used in practice because it retains algorithmic simplicity while offering an adequate degree of approximation. Newton-Cotes rules of higher order may be defined, but are more difficult to work with and thus are rarely used.

Through the use of tensor product principles, univariate Newton-Cotes quadrature schemes can be generalized for higher dimensional integration. Suppose one wishes to integrate a real-valued function defined on a rectangle $\{(x_1, x_2)|a_1 \leq x_1 \leq b_1, a_2 \leq x_2 \leq b_2\}$ in $\Re^2$. One way to proceed, is to compute the Newton-Cotes nodes and weights $\{(x_{1i}, w_{1i})|i = 1, 2, \ldots, n_1\}$ for the real interval $[a_1, b_1]$ and the Newton-Cotes and weights $\{(x_{2j}, w_{2j})|j = 1, 2, \ldots, n_2\}$ for the real interval $[a_2, b_2]$. The tensor product Newton-Cotes rule for the rectangle would be comprised of the $n = n_1 \cdot n_2$ grid points of the form $\{(x_{1i}, x_{2j})|i = 1, 2, \ldots, n_1; j = 1, 2, \ldots, n_2\}$ with associated weights $\{w_{ij} = w_{1i} \cdot w_{2j}|i = 1, 2, \ldots, n_1; j = 1, 2, \ldots, n_2\}$. This construction principle can be applied to an arbitrary dimension using repeated tensor product operations.

In most computational economic applications, it is not possible to determine a priori how many partition points are needed to compute an integral to a desired level of accuracy using a Newton-Cotes quadrature rule. One solution to this problem is to use an *adaptive quadrature* strategy whereby one increases the number of points at which the integrand is evaluated until the sequence of estimates of the integral converge. Efficient adaptive Newton-Cotes quadrature schemes are especially easy to implement. One simple, but powerful, scheme calls for the number of intervals to be doubled with each iteration. Because the new partition points include the partition points used in the previous iteration, the computational effort required to form the new integral estimate is cut in half. More sophisticated adaptive Newton-Cotes quadrature techniques relax the requirement that the intervals be equally

spaced and concentrate new evaluation points in those areas where the integrand appears to be most irregular.

5.2 Gaussian Quadrature

Gaussian quadrature rules are constructed with respect to specific weighting functions. Specifically, for a weighting function w defined on an interval $I \subset \Re$ of the real line, and for a given order of approximation n, the quadrature nodes $x_1, x_2, \ldots, x_n$ and quadrature weights $w_1, w_2, \ldots, w_n$ are chosen so as to satisfy the $2n$ 'moment-matching' conditions:

$$\int_I x^k w(x)\, dx = \sum_{i=1}^{n} w_i x_i^k, \text{ for } k = 0, \ldots, 2n-1.$$

Integral approximations are then formed using weighted sums of values of f at selected nodes:

$$\int_I f(x)w(x)\, dx \approx \sum_{i=1}^{n} w_i\ f(x_i).$$

Gaussian quadrature over a bounded interval with respect to the identity weighting function, $w \equiv 1$, is called Gauss-Legendre quadrature. Gauss-Legendre quadrature may be used to compute the area under a curve, and can easily be generalized to integration on higher dimensional spaces using tensor product principles. By construction, an n-point Gauss-Legendre quadrature rule will exactly compute the integral of any polynomial of order $2n-1$ or less. Thus, if f can be closely approximated by a polynomial, a Gauss-Legendre quadrature should provide an accurate approximation to the integral. Furthermore, Gauss-Legendre quadrature is consistent for Riemann integrable functions. That is, if f is Riemann integrable, then the approximation afforded by Gauss-Legendre quadrature can be made arbitrarily precise by increasing the number of nodes n.

Gauss-Legendre quadrature is the numerical integration method of choice when f possesses continuous derivatives, but should be applied with great caution otherwise. If the function f possesses known kink points, it is often possible to break the integral into the sum of two or more integrals of smooth functions. If these or similar steps do not produce smooth integrands,

Table 5.1: Errors for Selected Quadrature Methods

Function	Degree	Trapezoid Rule	Simpson Rule	Gauss-Legendre
$\exp(-x)$	10	1.36e+001	3.57e-001	8.10e-002
	20	3.98e+000	2.31e-002	2.04e-008
	30	1.86e+000	5.11e-003	1.24e-008
$(1+25x^2)^{-1}$	10	8.85e-001	9.15e-001	8.65e-001
	20	6.34e-001	6.32e-001	2.75e+001
	30	4.26e-001	3.80e-001	1.16e+004
$\lvert x\rvert^{0.5}$	10	7.45e-001	7.40e-001	6.49e-001
	20	5.13e-001	4.75e-001	1.74e+001
	30	4.15e-001	3.77e-001	4.34e+003

then Newton-Cotes quadrature methods may be more efficient than Gaussian quadrature methods because they limit the error caused by the kinks and singularities to the interval in which they occur.

When the weighting function w is the continuous probability density for some random variable $\tilde{X}$, Gaussian quadrature has a very straightforward interpretation. In this context, Gaussian quadrature essentially 'discretizes' the continuous random variable $\tilde{X}$ by constructing a discrete random variable with mass points x_i and probabilities w_i that approximates $\tilde{X}$ in the sense that both random variables have the same moments of order less than $2n$:

$$\sum_{i=1}^{n} w_i x_i^k = E(\tilde{X}^k) \text{ for } k = 0, \ldots, 2n-1.$$

Given the mass points and probabilities of the discrete approximant, the expectation of any function of the continuous random variable $\tilde{X}$ may be approximated using the expectation of the function of the discrete approximant, which requires only the computation of a weighted sum:

$$Ef(\tilde{X}) = \int f(x)\ w(x)\ dx \approx \sum_{i=1}^{n} f(x_i) w_i.$$

For example, the three-point Gauss-Hermite approximation to the standard univariate normal distribution $\tilde{Z}$ is characterized by the condition that its

The 0^{th} through 5^{th} moments match those of the standard normal: are $E\tilde{Z}^0 = 1$, $E\tilde{Z}^1 = 0$, $E\tilde{Z}^2 = 1$, $E\tilde{Z}^3 = 0$, $E\tilde{Z}^4 = 3$, and $E\tilde{Z}^5 = 0$. One can easily verify that these conditions are satisfied by a discrete random variable with mass points $x_1 = -\sqrt{3}$, $x_2 = 0$, and $x_3 = \sqrt{3}$ and associated probabilities w_1=1/6, $w_2 = 2/3$, and $w_3 = 1/6$.

Computing the n-degree Gaussian nodes and weights is a non-trivial task which involves solving the $2n$ nonlinear equations for $\{x_i\}$ and $\{w_i\}$. Efficient, specialized numerical routines for computing Gaussian quadrature nodes and weights are available for different weighting functions, including virtually all the better known probability distributions, such as the uniform, normal, gamma, exponential, Chi-square, and beta distributions. Gaussian quadrature with respect to the identity weight is called Gauss-Legendre quadrature; Gaussian quadrature with respect to normal probability densities is called Gauss-Hermite quadrature.[1]

As was the case with Newton-Cotes quadrature, tensor product principles may be applied to univariate Gauss-Hermite quadrature rules to develop quadrature rules for multivariate normal distributions. Suppose, for example, that $\tilde{X}$ is a d-dimensional normal random variable with mean vector μ and variance-covariance matrix Σ. Then $\tilde{X}$ is distributed as $\mu + \tilde{Z}R$ where R is the Cholesky square root of Σ (e.g., $\Sigma = R'R$) and $\tilde{Z}$ is a row d-vector of independent standard normal variates. If $\{z_i, w_i\}$ are the degree n Gaussian nodes and weights for a standard normal variate, then a n^d degree approximation for $\tilde{X}$ may be constructed using tensor products. For example, in two dimensions the nodes and weights would take the form

$$x_{ij} = (\mu_1 + R_{11}z_i + R_{21}z_j, \mu_2 + R_{12}z_i + R_{22}z_j)$$

and

$$p_{ij} = p_i p_j.$$

The Gaussian quadrature scheme for normal variates may also be used to develop a reasonable scheme for discretizing lognormal random variates. By definition, $\tilde{Y}$ is lognormally distributed with parameters μ and σ^2 if, and only if, it is distributed as $\exp(\tilde{X})$ were $\tilde{X}$ is normally distributed with mean μ and variance σ^2. It follows that if $\{x_i, w_i\}$ are Gauss-Hermite nodes and weights

[1]Most numerical analysis books use the term Gauss-Hermite quadrature to refer to the standard weighting function $w(x) = \exp(-x^2)$, which differs from the standard normal density only by a multiplicative constant of integration.

for a Normal(μ, σ^2) distribution, then $\{y_i, w_i\}$, where $y_i = \exp(x_i)$, provides a reasonable discrete approximant for a Lognormal(μ, σ^2) distribution. Given this discrete approximant for the lognormal distribution, one can estimate the expectation of a function of $\tilde{Y}$ as follows: $Ef(\tilde{Y}) = \int f(y)\ w(y)\ dy \approx \sum_{i=1}^{n} f(y_i) w_i$ This integration rule for lognormal distributions will be exact if f is a polynomial of degree $2n-1$ and less in $\log(y)$ (*not* in y).

5.3 Monte Carlo Integration

Monte Carlo integration methods are motivated by the Strong Law of Large Numbers. One version of the Law states that if $x_1, x_2, \ldots$ are independent realizations of a random variable $\tilde{X}$ and f is a continuous function, then

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} f(x_i) = Ef(\tilde{X})$$

with probability one.

The Monte Carlo integration scheme is thus a simple one. To compute an approximation to the integral of $f(\tilde{X})$, one draws a random sample $x_1, x_2, \ldots, x_n$ from the distribution of $\tilde{X}$ and sets

$$Ef(\tilde{X}) \approx \frac{1}{n} \sum_{i=1}^{n} f(x_i).$$

Matlab offers two intrinsic random number generators. The routine `rand` generates a random sample from the uniform(0,1) distribution stored in either vector or matrix format. Similarly, the routine `randn` generates a random sample from the standard normal distribution stored in either vector or matrix format. In particular, a call of the form `x=rand(n)` or `x=randn(n)` generates a random sample of n realizations and stores it in a row vector.

The uniform random number generator is useful for generating random samples from other distributions. Suppose $\tilde{X}$ has a cumulative distribution function

$$F(x) = \Pr(\tilde{X} \leq x)$$

whose inverse has a well-defined closed form. If $\tilde{U}$ is uniformly distributed on $(0, 1)$, then

$$F^{-1}(\tilde{U})$$

is distributed as $\tilde{X}$. Thus, to generate a random sample $x_1, x_2, \ldots, x_n$ from the $\tilde{X}$ distribution, one generates a random sample $u_1, u_2, \ldots, u_n$ from the uniform distribution and sets $x_i = F^{-1}(u_i)$.

The standard normal random number generator is useful for generating random samples from related distributions. For example, to generate a random sample of n lognormal variates, one may use the script

```
x = exp(mu+sigma*randn(n));
```

where `mu` and `sigma` are the mean and standard deviation of the distribution. To generate a random sample of n d-dimensional normal variates one may use the script

```
x = randn(n,d)*chol(Sigma)+mu(ones(n,1),:);
```

where `Sigma` is the d by d variance-covariance matrix and `mu` is the mean vector in row form.

A fundamental problem that arises with Monte Carlo integration is that it is almost impossible to generate a truly random sample of variates for any distribution. Most compilers and vector processing packages provide intrinsic routines for computing so-called random numbers. These routines, however, employ iteration rules that generate a purely deterministic, not random, sequence of numbers. In particular, if the generator is repeatedly initiated at the same point, it will return the same sequence of 'random' variates each time. About all that can be said of numerical random number generators is that good ones will generate sequences that appear to be random, in that they pass certain statistical tests for randomness. For this reason, numerical random number generators are often more accurately said to generate sequences of 'pseudo-random' rather than random numbers.

Monte Carlo integration is easy to implement and may be preferred over Gaussian quadrature if the a routine for computing the Gaussian mass points and probabilities is not readily available or if the integration is over many dimensions. Monte Carlo integration, however, is subject to a sampling error that cannot be bounded with certainty. The approximation can be made more accurate, in a statistical sense, by increasing the size of the random sample, but this can be expensive if evaluating f or generating the pseudo-random variate is costly. Approximations generated by Monte Carlo integration will vary from one integration to the next, unless initiated at the same point, making the use of Monte Carlo integration in conjunction within other iterative schemes, such as dynamic programming or maximum likelihood estimation, problematic. So-called quasi Monte-Carlo methods can circumvent some of the problems associated with Monte-Carlo integration.

5.4 Quasi-Monte Carlo Integration

Although Monte-Carlo integration methods originated using insights from probability theory, recent extensions have severed that connection and, in the process, demonstrated ways in which the methods can be improved. Monte-Carlo methods rely on sequences $\{x_i\}$ with the property that

$$\lim_{n\to\infty} \frac{b-a}{n} \sum_{i=1}^{\infty} f(x_i) = \int_a^b f(x)\, dx.$$

Any sequence that satisfies this condition for arbitrary (Riemann) integrable functions can be used to approximate an integral on $[a, b]$. Although the Law of Large Numbers assures us that this is true when the x_i are independent and identically distributed random variables, other sequences also satisfy this property. Indeed, it can be shown that sequences that are explicitly non-random, but instead attempt to fill in space in a regular manner exhibit improved convergence properties.

There are numerous schemes for generating equidistributed sequences. The best known are the Neiderreiter, Weyl, and Haber. The following Matlab script generates equidistributed sequences of length n for the unit hypercube:

```
eds_pp=sqrt(primes(7920));
i=(1:n)';
switch upper(type(1))
  case 'N'                  % Neiderreiter
    j=2.^((1:d)/(d+1));
    x=i*j;
    x=x-fix(x);
  case 'W'                  % Weyl
    j=eds_pp(1:d);
    x=i*j;
    x=x-fix(x);
  case 'H'                  % Haber
    j=eds_pp(1:d);
    x=(i.*(i+1)./2)*j;
    x=x-fix(x);
end
```

The Matlab toolbox accompanying the textbook includes a function `qnwequi` that generates the equidistributed nodes for integration over an arbitrary bounded interval in a space of arbitrary dimension. The calling sequence takes the form

```
[x,w] = qnwequi(n,a,b,type);
```

where `x` are the nodes, `w` are the weights, `n` is the number nodes and weights, `a` is the vector of left endpoint, `b` is the vector of right endpoints, and `type` refers to the type of equidistributed sequence ('N'-Neiderrieter, 'W'-Weyl, and 'H'-Haber). For example, suppose one wished to compute the integral of $\exp(x_1 + x_2)$ over the rectangle $[1, 2] \times [0, 5]$ in $\Re^2$. On could invoke `qnwequi` to generate a sequence of, say, 1000 equidistribued Neiderrieter nodes and weights and form the weighted sum:

```
[x,w] = qnwequi(1000,[1 0],[2 5],'N');
integral = w'*exp(x(:,1)+x(:,2));
```

Two-dimensional examples of the sequences generated by `qnwequi` are illustrated in Figure 5.1. Each of the plot shows $16,000$ values. It is evident that the Neiderreiter and Weyl sequences are very regular, showing far less blank space than the Haber sequence or the pseudo-random sequence. This demonstrates that it is possible to have sequences that are not only uniformly distributed in an *ex ante* or probabilistic sense but also in an *ex post* sense, thereby avoiding the clumpiness exhibited by truly random sequences.

Figure 5.2 demonstrates how increasing the number of points in the Neiderreiter sequence progressively fills in the unit square.

To illustrate the quality of the approximations, Table 5.2 displays the approximation error for the integral

$$\int_{-\infty}^{0} \int_{-\infty}^{0} \exp\left(-\tfrac{1}{2} x_1^2 x_2^2\right) \, dx_1 \, dx_2,$$

the solution of which is $\pi/2$. It is clear that the method requires many evaluation points for even modest accuracy and that large increases in the number of points reduces the error very slowly.[2]

5.5 Numerical Differentiation

The most natural way to approximate a derivative is to replace it with a finite difference. The definition of a derivative,

$$f'(x) = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h},$$

[2]Part of the problem may be due to truncation of the domain of integration to $[-8, 0] \times [-8, 0]$.

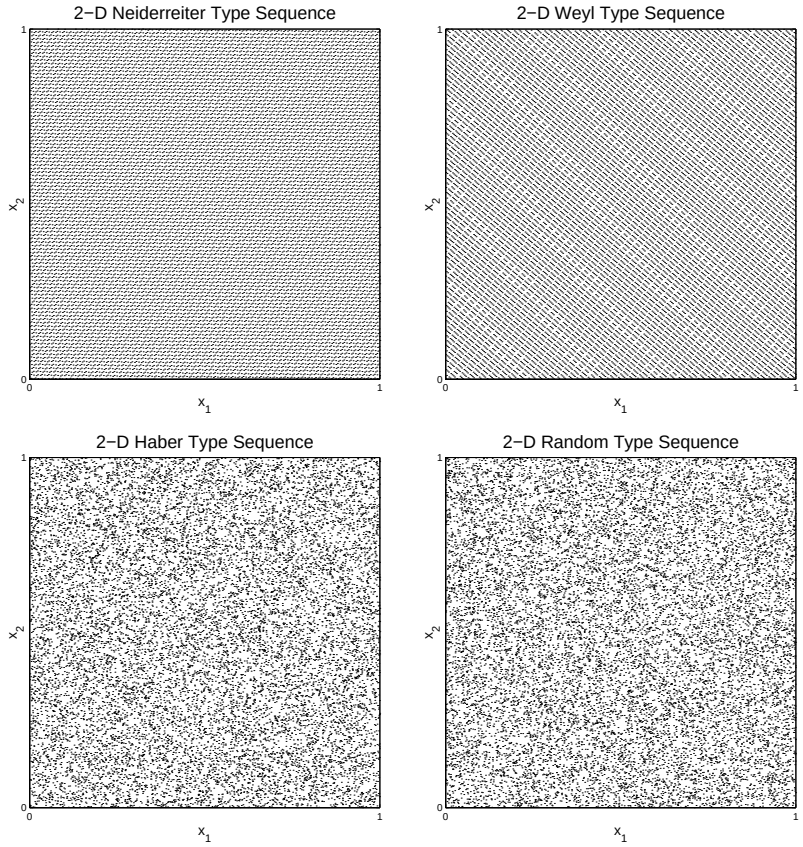

Figure 5.1: Alternative Equidistributed Sequences

Table 5.2: Approximation Errors for Alternative Quasi-Monte Carlo Methods

n	Neiderreiter	Weyl	Haber	Pseudo Random
1000	0.08533119	0.03245903	0.08233608	0.21915134
10000	0.01809421	0.00795709	0.00089792	0.01114914
100000	0.00110185	0.00051383	0.00644085	0.01735175
250000	0.00070244	0.00010050	0.00293232	0.00157189

suggests a natural way to do this. One can simply take h to be a small number, knowing that, for h small enough, the error of the approximation will also be small. On first blush, one may be tempted to pick an h as small as

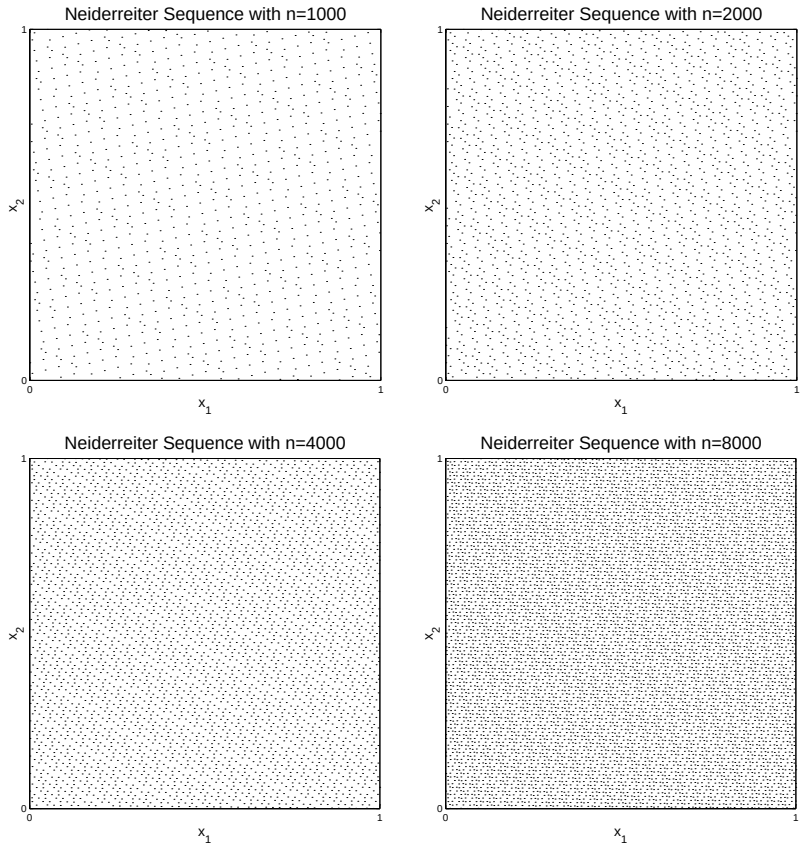

Figure 5.2: Fill in of the Neiderreiter Sequence

possible, the machine infinitesimal. However, too small a choice of h renders the approximation susceptible to rounding error. The selection of the step h is thus a nontrivial matter.

The study of finite difference approximations can be put on a firm basis using Taylor approximations. We know, for example, that

$$f(x+h) = f(x) + f'(x)h + O(h^2),$$

where $O(h^2)$ means that other terms in the expression are expressible in terms of second or higher powers of h. If we rearrange this expression we see that

$$f'(x) = [f(x+h) - f(x)]/h + O(h).$$

(since $O(h^2)/h = O(h)$), so that the approximation to the derivative $f'(x)$ that has an $O(h)$ error. This is the simplest form of a forward difference approximation; the "forward" referring to the fact that we approximate f' by evaluating at x and $x+h$, where h is some positive amount. The analogous backward difference approximation

$$f'(x) = [f(x) - f(x-h)]/h + O(h)$$

also offers an approximation of order $O(h)$.

Consider now the two second order Taylor expansions:

$$f(x+h) = f(x) + f'(x)h + f''(x)\frac{h^2}{2} + O(h^3)$$

and

$$f(x-h) = f(x) - f'(x)h + f''(x)\frac{h^2}{2} + O(h^3).$$

If we subtract one of these from the other, rearrange, and divide by h, we get

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2).$$

This is the simplest of the centered finite difference approximations. Its error is $O(h^2)$. The centered finite difference approximation is thus theoretically more accurate than either one-sided approximation.

Difference approximations for higher order derivatives can be found using the same approach. For our purposes a centered difference approximation to the second derivative will suffice. Again, we start with a Taylor approximation, this time of third order:

$$f(x+h) = f(x) + f'(x)h + f''(x)\frac{h^2}{2} + f^{(3)}(x)\frac{h^3}{6} + O(h^4)$$

and

$$f(x-h) = f(x) - f'(x)h + f''(x)\frac{h^2}{2} - f^{(3)}(x)\frac{h^3}{6} + O(h^4)$$

Adding these together cancels the odd ordered terms. Rearranging we get

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + O(h^2).$$

Thus we have centered difference approximations to both $f'(x)$ and $f''(x)$ that have $O(h^2)$ errors.

The second problem is how h should be chosen. As with convergence criteria, there is no one rule that always works. If h is made too small, round-off error can make the results meaningless. On the other hand, too large an h provides a poor approximation, even if exact arithmetic is used. This is illustrated in Figure 5.3, which displays the errors in approximating the derivative of $\exp(x)$ at $x = 1$ as a function of h. The approximation improves as h is reduced to the point that it is approximately equal to $\sqrt{\epsilon}$ (the square root of the machine precision), shown as a star on the horizontal axis. Further reductions in h actually worsen the approximation because of the inaccuracies due to inexact arithmetic. This gives credence to the rule of thumb that, for one-sided approximations, h should be chosen to be of size $\sqrt{\epsilon}$ relative to x. When x is small, however, it is better not to let h get too small. We suggest the rule of thumb of setting

$$h = \max(x, 1)\sqrt{\epsilon}.$$

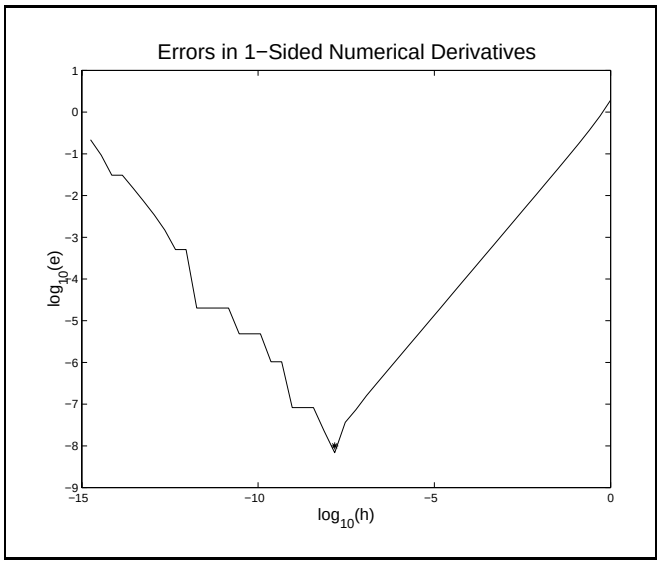

Figure 5.3

Figure 5.4 shows an analogous plot for two-sided approximations. It is evident that the error is minimized at a much higher value of h, at approxi-

mately $\sqrt[3]{\epsilon}$. A good rule of thumb is to set

$$h = \max(x, 1)\sqrt[3]{\epsilon}$$

when using two-sided approximations.

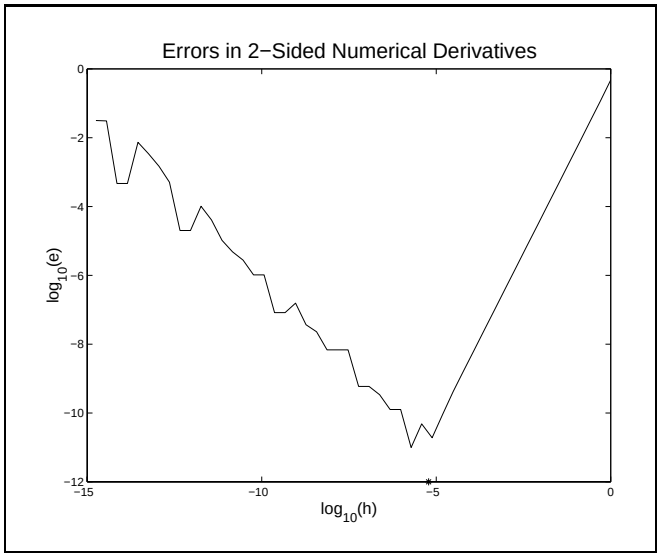

Figure 5.4

There is a further, and more subtle, problem. If $x + h$ cannot be represented exactly but is instead equal to $x + h + e$, then we are actually using the approximation

$$\begin{array}{rcl} \frac{f(x+h+e)-f(x+h)}{e}\frac{e}{h} + \frac{f(x+h)-f(x)}{h} & \approx & f'(x+h)\frac{e}{h} + f'(x) \\ & \approx & \left(1+\frac{e}{h}\right) f'(x). \end{array}$$

Even if the rounding error e is on the order of machine accuracy, ϵ, and h on the order of $\sqrt{\epsilon}$, we have introduced an error on the order of $\sqrt{\epsilon}$ into the calculation. It is easy to deal with this problem, however. Letting `xh` represent $x + h$, define h in the following way:

```
h=sqrt(eps)*max(x,1); dh=x+h; h=dh-x;
```

The function below computes the two-sided finite difference approximation for the Jacobian of an arbitrary function. For a real-valued function with an n-vector input, the output is an $m \times n$ matrix:

```
function fjac = fdjac(f,x);
h = eps^(1/3)*max(abs(x),1);
for j=1:length(x);
   x1 = x; x1(j) = x(j) + h(j);
   x0 = x; x0(j) = x(j) - h(j);
   fjac(:,j) = (feval(f,x1)-feval(f,x0))/(x1(j)-x0(j));
end
```

One further difficulty arises in the use of finite difference approximations. Sometimes it is not possible or convenient to pick the values at which the function is approximated. Instead, we start with a set of function values and require derivative approximations given the known values. Such situations commonly arise in solving partial differential equations in which we desire $O(h^2)$ accuracy.

If the points are evenly spaced we can use the two-sided formulas already present and no particular difficulties arise. In some cases, however, it is difficult to use centered difference approximations, for example when we require an approximation at a boundary of the domain of a function. Suppose that you are approximating a function over the range $[a, b]$ and need to have approximations to $f'(a)$ or $f'(b)$. There are two recommended approaches to this problem, neither of which is to use $O(h)$ forward and backward difference approximations. If we want to preserve the $O(h^2)$ errors of the approximation, one alternative is to obtain better forward and backward approximations. To see how to accomplish this consider a Taylor expansion of $f(x + h)$ and $f(x + 2h)$ at $f(x)$:

$$f(x + h) = f(x) + f'(x)h + \tfrac{1}{2} f''(x)h^2 + O(h^3)$$

and

$$f(x + 2h) = f(x) + f'(x)2h + f''(x)2h^2 + O(h^3),$$

and subtract the latter from 4 times the former (to eliminate the h^2 term). Upon rearranging we get

$$f'(x) = \frac{1}{2h}\Big(-3f(x) + 4f(x + h) - f(x + 2h)\Big) + O(h^2).$$

Thus we have a forward difference approximation with $O(h^2)$ errors that can be used at the lower boundary of the domain of f. The analogous $O(h^2)$ backward difference approximation for an upper boundary is

$$f'(x) = \frac{1}{2h}\Big(f(x - 2h) - 4f(x - h) + 3f(x)\Big) + O(h^2).$$

To aid implementation of finite difference methods, we have provided a Matlab function, FDOP, in the DE toolbox to compute matrix finite difference operators. A call to `D1=FDOP(dx,n,1)` will return an $n \times n$ matrix such that $D1 * V$, where V is an n-vector of function values, will approximate the associated values of the first derivative of V. Similarly `D2=FDOP(dx,n,2)` can be used to approximate the values of the second derivative. The first and last rows of the operators provide one sided approximations at the endpoints, but these can be replaced by appropriate boundary information.

Before we leave the subject of finite difference approximations, we should note that we may encounter situations in which the forward step and the backward step are of different sizes, say h and λh, respectively. To handle this situation, we proceed as before by taking Taylor expansions at $x+h$ and $x - \lambda h$:

$$f(x+h) = f(x) + f'(x)h + f''(x)\frac{h^2}{2} + O(h^3).$$

and

$$f(x-\lambda h) = f(x) - f'(x)\lambda h + f''(x)\frac{\lambda^2 h^2}{2} + O(h^3).$$

This time, however, we need to subtract the second expression from λ^2 times the first:

$$\lambda^2 f(x+h) - f(x-\lambda h) = (\lambda^2 - 1)f(x) + (\lambda^2 + \lambda)f'(x)h + O(h^3)$$

and solve for $f'(x)$:

$$f'(x) = \frac{1}{h}\left(\frac{\lambda}{\lambda+1}f(x+h) + \frac{1-\lambda}{\lambda}f(x) - \frac{1}{\lambda(\lambda+1)}f(x-\lambda h)\right) + O(h^2).$$

Thus we can obtain an $O(h^2)$ approximation even when the points are unevenly spaced. The approximation comes at a cost: it takes three evaluation points rather than two. Only when $\lambda = 1$ (even spacing) does the central term drop out and we are back to the simple formula previously derived.

Using the methods just described, one can derive the following $O(h^2)$ approximations:

$$f'(x) = \frac{1}{h}\left(-\frac{2+\lambda}{1+\lambda}f(x) + \frac{1+\lambda}{\lambda}f(x+h) - \frac{1}{\lambda(1+\lambda)}f\left(x+(1+\lambda)h\right)\right) + O(h^2)$$

$$f'(x) = \frac{1}{h}\left(\frac{1}{\lambda(1+\lambda)}f\left(x-(1+\lambda)h\right) - \frac{1+\lambda}{\lambda}f(x-h) + \frac{2+\lambda}{1+\lambda}f(x)\right) + O(h^2)$$

and

$$f"(x) = \frac{2}{h^2}\left(\frac{1}{\lambda+1}f(x+h) - \frac{1}{\lambda}f(x) + \frac{1}{\lambda(\lambda+1)}f(x-\lambda h)\right) + O(h^2).$$

The first is the forward difference approximation for the unevenly spaced points x, $x+h$ and $x+(1+\lambda)h$ ($\lambda = 1$ again represents even spacing). The second is the analogous backward approximations for the points $x-(1+\lambda)h$, $x-h$ and x.[3] The last expression is the centered finite difference approximation to the second derivative for unevenly spaced points $x-\lambda h$, x and $x+h$. The various finite difference approximations are summarized in Table 5.3.

5.6 An Integration Toolbox

The Matlab toolbox accompanying the textbook includes a pair of functions `qnwtrap` and `qnwsimp` that generate the trapezoid and Simpson's rule quadrature nodes and weights for integration over an arbitrary bounded interval on the real line. The calling sequences take the form

```
[x,w] = qnwtrap(n,a,b);
```

and

```
[x,w] = qnwsimp(n,a,b);
```

where `x` are the nodes, `w` are the weights, `n` is the number nodes and weights, `a` the left endpoint, and `b` is the right endpoint. For example, to compute the definite integral of $\exp(x)$ on $[-1,2]$ using a 21 point trapezoid rule one would write:

```
[x,w] = qnwtrap(21,-1,2);
integral = w'*exp(x);
```

[3]The forward formula can be seen to be identical to the centered formula is we set $\lambda = -2$. The backward formula is not a special case, however, unless we also allow h to be negative. Rather than confuse the issue, it seems easier to work with the appropriate formula and keep both h and λ positive.

Table 5.3: Finite Difference Formulae with $O(h^2)$ Errors

Centered

$$f'(x) = \frac{1}{h}\left(\frac{\lambda}{\lambda+1}f(x+h) + \frac{1-\lambda}{\lambda}f(x) - \frac{1}{\lambda(\lambda+1)}f(x-\lambda h)\right) + O(h^2).$$

$$f"(x) = \frac{2}{h^2}\left(\frac{1}{\lambda+1}f(x+h) - \frac{1}{\lambda}f(x) + \frac{1}{\lambda(\lambda+1)}f(x-\lambda h)\right) + O(h^2).$$

Forward

$$f'(x) = \frac{1}{h}\left(-\frac{2+\lambda}{1+\lambda}f(x) + \frac{1+\lambda}{\lambda}f(x+h) - \frac{1}{\lambda(1+\lambda)}f\left(x+(1+\lambda)h\right)\right) + O(h^2)$$

Backward

$$f'(x) = \frac{1}{h}\left(\frac{1}{\lambda(1+\lambda)}f\left(x-(1+\lambda)h\right) - \frac{1+\lambda}{\lambda}f(x-h) + \frac{2+\lambda}{1+\lambda}f(x)\right) + O(h^2)$$

$\lambda = 1$ gives the evenly spaced formulae

In this example, the trapezoid rule yields an estimate that is accurate to two significant digits. The Simpson's rule with the same number of nodes yields an estimate that is accurate to five significant digits.

The Matlab functions `qnwtrap` and `qnwsimp` use tensor products to generate Newton-Cotes nodes and weights for integration over an arbitrary bounded interval $[a, b]$ in higher dimensional spaces. For example, suppose one wished to compute the integral of $\exp(x_1 + x_2)$ over the rectangle $[1, 2] \times [0, 5]$ in $\Re^2$. On could invoke `qnwtrap` to construct a grid of, say, 2601 quadrature nodes produces by taking the cross-product of 51 nodes in the x_1 direction and 51 nodes in the x_2 direction:

```
[x,w] = qnwtrap([51 51],[1 0],[2 5]);
integral = w'*exp(x(:,1)+x(:,2));
```

Application of the trapezoid rule in this example yields an estimate of 689.1302, which is accurate to three significant digits; application of Simpson's rule with the same number of nodes yields an estimate of 688.5340, which is accurate to six significant digits.

Matlab offers two Newton-Cotes quadrature routines, `quad` and `quad8` both of which employ an adaptive Simpson's rule.

The Matlab toolbox accompanying the textbook includes a function `qnwlege` that generates the Gauss-Legendre quadrature nodes and weights for integration over an arbitrary bounded interval on the real line. The calling sequence takes the form

```
[x,w] = qnwlege(n,a,b);
```

where `x` are the nodes, `w` are the weights, `n` is the number nodes and weights, `a` the left endpoint, and `b` is the right endpoint. For example, to compute the definite integral of $\exp(x)$ on $[-1, 2]$ using a 21 point rule one would write

```
[x,w] = qnwlege(21,-1,2);
integral = w'*exp(x);
```

In this example, Gauss-Legendre quadrature produces an estimate that is accurate to fourteen significant digits, eight more than Simpson's quadrature with the same number of nodes.

The Matlab function `qnwlege` is designed to generate Gauss-Legendre nodes and weights over an arbitrary bounded interval in higher dimensional spaces. The nodes and weights are constructed using tensor products of the nodes and weights of the one-dimensional Gauss-Legendre nodes and weights. For example, suppose one wished to compute the integral of $\exp(x_1 + x_2)$ over

the rectangle $[1,2] \times [0,5]$ in $\Re^2$. On could invoke `qnwlege` to construct a grid of, say, 20 Gaussian quadrature nodes, derived from the cross-product of 5 nodes in the x_1 direction and 4 nodes in the x_2 direction, and then form the weighted sum of the assigned weights and function values at the nodes:

```
[x,w] = qnwlege([5 4],[1 0],[2 5]);
integral = w'*exp(x(:,1)+x(:,2));
```

This computation would yield an approximate answer of 688.5323, which is very close to the correct answer 688.5336 and more accurate than the approximation afforded by Simpson's rule using nearly 100 times more function evaluations.

The Matlab toolbox accompanying the textbook also includes a function `qnwnorm` that generates the Gauss-Hermite quadrature nodes and weights for computing the expectations of functions of normal random variates. For univariate normal distributions, the calling sequence takes the form

```
[x,w] = qnwnorm(n,mu,var);
```

where `x` are the nodes, `w` are the probability weights, `n` is the number nodes and weights, `mu` the mean of the distribution, and `var` is the variance of the distribution. If `mu` and `var` are omitted, the mean and variance are assumed to be 0 and 1, respectively. For example, suppose one wanted to compute the expectation of $\exp(\tilde{X})$ where $\tilde{X}$ is normally distributed with mean 2 and variance 4. An approximate expectation could be computed using the following Matlab code:

```
[x,w] = qnwnorm(3,2,4);
expectation = w'*exp(x);
```

The Matlab function `qnwnorm` is designed to generate Gauss-Hermite nodes and weights for multivariate normal random variables. For example, suppose one wished to compute the expectation of, say, $\exp(\tilde{X}_1 + \tilde{X}_2)$ where $\tilde{X}_1$ and $\tilde{X}_2$ are jointly normal with mean vector $[3; 4]$ and variance covariance matrix $[2 - 1; -14]$. One could invoke `qnwnorm` to construct a grid of 100 Gaussian quadrature nodes as the cross-product 10 knots in the x_1 direction and 10 knots in the x_2 direction, and then form the weighted sum of the assigned weights and function values at the nodes:

```
[x,w] = qnwnorm([10 10],[3;4],[2 -1; -1 4]);
integral = w'*exp(x(:,1)+x(:,2));
```

This computation would yield an approximate answer of 8103.083, which is accurate to 7 significant digits.

The Matlab toolbox accompanying the textbook provides a function `qnwlogn` that in this manner generates quadrature nodes and weights for computing the expectations of functions of lognormal random variate. For univariate lognormal distributions, the calling sequence takes the form

```
[x,w] = qnwlogn(n,mu,var);
```

which in Matlab may be implemented as follows:

```
[x,w] = qnwlogn(n,mu,var);
expectation = w'*f(x);
```

5.7 Initial Value Problems

Differential equations pose the problem of inferring a function given information about its derivatives and additional 'boundary' conditions. Differential equations may characterized as either ordinary differential equations (ODEs), whose solutions are functions of a single argument, and partial differential equations (PDEs), whose solutions are functions of multiple arguments. Both ODEs and PDEs may be solved numerically using finite difference methods.

From a numerical point of view the distinction between ODEs and PDEs is less important than the distinction between initial value problems (IVPs), which can be solved in a recursive or evolutionary fashion, and boundary value problems (BVPs), which require the entire solution to be computed simultaneously because the solution at one point (in time and/or space) depends on the solution everywhere else. With IVPs, the solution is known at some point or points and the solution near these points can then be (approximately) determined. This, in turn, allows the solution at still other points to be approximated and so forth. BVPs, on the other hand, require simultaneous solution of the differential equation and the boundary conditions. We take up the solution of IVPs in this section, but defer discussion of BVPs for a later Chapter.

The most common initial value problem is to find a function $x : [0, T] \subset \Re \mapsto \Re^d$ whose initial value $x(0)$ is known and which, over its domain, satisfies the differential equation

$$x(t)' = f(x(t), t).$$

Here, x is a function of a scalar t, time, and $f : \Re^{d+1} \mapsto \Re$ is a given function. Many problems in economics are time-autonomous, in which case the differential equation takes the form

$$x(t)' = f(x(t)).$$

Although the differential equation contains no derivatives of order higher than one, the equation is more general than it might at first appear, because higher order derivatives can always be eliminated by expanding the number of variables. For example, consider the second order differential equation

$$y(t)'' = f(y(t), y(t)', t).$$

If z is the first derivative of x, so that $z' = x''$, then the differential equation may be written in first order form

$$y' = z$$

$$z' = f(y, z, t).$$

A solution to a system of differential equations is a vector-valued function $x(t)$ (of dimension d) that satisfies the differential equation subject to a set of boundary conditions. In initial value problems, the boundary conditions define the values of the variables at a single point in time. This allows initial value problems to be solved using an iterative procedure. First the direction of motion is calculated based on the current position of the system and a small step is taken in that direction. This is then repeated as many times as is desired. The inputs needed for these methods are the functions defining the system, f, an initial value, x_0, the time step size, h, and the number of steps to take, n.

The most simple form of such a procedure is Euler's method. The i^{th} iteration of the procedure generates an approximation for the value of the solution function x at time t_i

$$x_{i+1} = x_i + hf(x_i, t_i),$$

with the procedure beginning at the prescribed $x_0 = x(0)$. This method is fine for rough approximations if the time step is small enough. However, for many applications, the technique can give unacceptable results.

There are numerous refinements on the Euler method, the most commonly used being Runge-Kutta methods. Runge-Kutta methods are actually a class

of methods characterized by an order of approximation and by selection of certain key parameters. The derivation of these methods is fairly tedious for high order methods but are easily demonstrated for a second order model.

Runge-Kutta methods are based on Taylor approximations at a given starting point t.

$$x(t+h) = x + hf(x,t) + \frac{h^2}{2}\left(f_t + f_x f\right) + O(h^3),$$

where $x = x(t)$, $f = f(x,t)$ and f_x and f_t are the partial derivatives of f evaluated at (x,t). This equation could be used directly but would require obtaining explicit expressions for the partial derivatives f_x and f_t. A method that relies only on function evaluations is obtained by noting that

$$f(x+\lambda hf, t+\lambda h) \approx f + \lambda h\left(f_t + f_x f\right).$$

Substituting this into the previous expression yields

$$x(t+h) \approx x + h\Big[(1-\frac{\lambda}{2})f + \frac{\lambda}{2} f\Big(x+\lambda hf, t+\lambda h\Big)\Big].$$

Two simple choices for λ are $\frac{1}{2}$ and 1 leading to the following second order Runge-Kutta methods:

$$x(t+h) \approx x + hf\left(x + \frac{h}{2}f, t+\frac{h}{2}\right)$$

and

$$x(t+h) \approx x + \frac{h}{2}f + f\left(x+hf, t+h\right).$$

It can be shown that an optimal choice (in the sense of minimizing the truncation error) is to set $\lambda = 2/3$ (see Atkinson, pp. 369-370).

Further insight can be gained into the Runge-Kutta methods by relating them to Newton-Cotes numerical integration methods. In general

$$x(t+h) = x(t) + \int_t^{t+h} f(x(\tau),\tau)d\tau$$

Suppose that the integral in this expression is approximated used the trapezoid rule:

$$x(t+h) = x(t) + \frac{h}{2}\Big(f(x(t),t) + f\Big(x(t+h), t+h\Big)\Big).$$

Now use Euler's method to approximate the $x(t+h)$ term that appears on the right-hand side to obtain

$$x(t+h) = x(t) + \frac{h}{2}\Big(f(x(t),t) + f\Big(x(t) + hf(t,x(t)), t+h\Big)\Big),$$

which is the same formula as above with $\lambda = 1$. Thus combining two first order methods, Euler's method and the trapezoid method, results in a second order Runge-Kutta method.

The most widely used Runge-Kutta method is the classical fourth-order method. A derivation of this approach is tedious but the algorithm is straightforward:

$$x(t+h) = x + (F_1 + 2(F_2 + F_3) + F_4)/6,$$

where

$$F_1 = hf(x,t)$$

$$F_2 = hf(x + \tfrac{1}{2}F_1, t + \tfrac{1}{2}h)$$

$$F_3 = hf(x + \tfrac{1}{2}F_2, t + \tfrac{1}{2}h)$$

$$F_4 = hf(x + F_3, t + h).$$

It can be shown that the truncation error in any order k Runge-Kutta method is $O(h^{k+1})$. Also, just as a second order method can be related to the trapezoid rule for numerical integration, the fourth order Runge-Kutta method can be related to Simpson's rule (we leave this as an exercise).

The Matlab function RK4 is written to return an approximate solution $x(T)$ to $x' = f(x,t)$, s.t. $x(T(1)) = x0$, where T is a vector of values. Furthermore, it is designed to compute solutions for multiple initial values. If $x0$ is $d \times k$, RK4 will return a row vector of length dk for each time step. This contrasts with the internal Matlab ODE functions, which will process only a single initial value and therefore must be called within a loop to generate a phase diagram. Avoiding this loop results in much faster execution when a large set of trajectories are computed. To take advantage of this feature, however, the function that is passed to RK4 that defines the differential equation must be able to return a $d \times k$ matrix when its second input argument is a $d \times k$ matrix (see fishsys.m below for an illustration of how this is done).

There are numerous other approaches and refinements to solving initial value problems. Briefly, these include so-called multi-step algorithms which utilize information from previous steps to determine the current step direction (Runge-Kutta and Taylor series approaches are single-step methods). Also, any method can adapt the step size to the current behavior of the system by monitoring the truncation error, reducing (increasing) the step size if this error is unacceptably large (small). Adaptive schemes are important if one requires a given level of accuracy.[4]

As an example of an initial value problem, consider the model of a commercial fishery developed by V.L. Smith (JPE 77 #2 (1969): 181-198). The model is interesting in that it makes fairly simple behavioral assumptions but results in a complex 2-variable system of first order nonlinear differential equations. The model first determines a short-run (instantaneous) equilibrium given the current size of the fish stock and the size of the fishing industry. This equilibrium is determined by the demand for fish and a fishing firm profit function, which together determine the short-run equilibrium catch rate and firm profit level. The model's dynamic behavior is governed by a growth rate for the fish stock and a rate of entry into the fishing industry. The former depends on the biological growth of the fish population and on the current catch rate, whereas the later depends on the current profitability of fishing. Taken together, these determine the adjustment process of the fish stock and the industry size.

The model is summarized below:[5]
Equations:

$p = \alpha - \beta K y$	inverse demand for fish
$\pi = py - cy^2/2S - f$	profit function of representative fishing firm
$S' = (a - bS)S - Ky$	fish population dynamics
$K' = \delta\pi$	entry/exit from industry

[4]The Matlab functions ODE23 and ODE45 are implemented in this way, with ODE45 a fourth order method.

[5]We have made slight changes to Smith's notation and simplified his model by making the industry adjustments occur at the same rate for positive and negative profits.

Variables:

p	the price of fish
K	size of the industry
y	catch rate of the representative firm
π	profit of the representative firm
S	fish population

Parameters:

$\alpha, \beta, c, f, a, b$ and δ

In this model K represents the size of the fishing industry, which is treated as a continuous variable. Smith called this the number of firms in the industry, but it is more accurately thought of as the total capital stock in use. By increasing effort, the catch rate per unit of capital, y, can be increased. The total supply (total catch rate) is defined as $Y = Ky$. Marginal cost is constant in units of capital but are quadratically increasing in the catch rate and inversely related to the total stock of fish. The parameter f represents a fixed cost per unit of capital. The biological process governing fish stocks depends on a recharging rate (a) and a mortality factor (b), the latter due to the stock approaching the biological carrying capacity (a/b).

The industry is competitive in the sense that catch rates are chosen by setting marginal cost equal to price:

$$p = cy/S.$$

This relationship, which can be thought of as the short-run inverse supply function per unit of capital. The short-run (market-clearing) equilibrium is determined by equating demand and supply:

$$\alpha - \beta K y = cy/S,$$

yielding a short-run equilibrium catch rate:

$$y = \alpha S/(c + \beta SK),$$

price

$$p = \alpha c/(c + \beta SK),$$

and profit function

$$\pi = \frac{c\alpha^2 S}{2(c+\beta SK)^2} - f.$$

All of these relationships are functions of the industry size and the stock of fish.

These results can be used to derive expressions for the time rates of change of the fish stock, which depends on the catch rate and of the industry size, which depends on current profits. The capital stock adjustment process is myopic, as it depends only on current profitability and not on expected future profitability (see Conrad and Clarke for discussion). The result is a 2 dimensional system of nonlinear differential equations:

$$\begin{aligned} S' &= (a-bS)S - \frac{\alpha SK}{c+\beta SK} \\ K' &= \delta\left(\frac{c\alpha^2 S}{2(c+\beta SK)^2} - f\right) \end{aligned}$$

The behavior of the model is more transparent if it is scaled by setting

$$\alpha = a = b = c = 1,$$

(there are four scaling measures: money, fish quantity, industry size and time).

$$\begin{aligned} S' &= (1-S)S - \tfrac{SK}{1+\beta SK} \\ K' &= \delta\left(\tfrac{S}{2(1+\beta SK)^2} - f\right) \end{aligned}$$

The system can be used to determine the behavior of the model starting at any initial stock level and industry size.

A useful device for summarizing the behavior of a dynamic system is the phase diagram, which shows the movement of the system for selected starting values; these curves are known as the trajectories. A phase diagram for this model with parameter values $\beta = 2.75$, $f = 0.06$ and $\delta = 10$ is exhibited in Figure **??**.

Phase diagrams typically include another set of curves besides the trajectories. The zero-isoclines are the points in the state space for which one

of the variables' time rate of change is zero. The intersections of the zero-isoclines are the system equilibria. In the fishery model, the zero-isocline for S is the set of points satisfying $S' = 0$; solving for K yields

$$K = \frac{1 - S}{1 - \beta(1 - S)S}.$$

($S' = 0$ is also satisfied whenever $S = 0$ thereby preventing fish stocks from becoming negative). Notice that this curve depends only on the parameter β. To find the K zero-isocline, set $K' = 0$, which results in a quadratic in K:

$$K^2 + \frac{2}{\beta S}K + \left(\frac{1}{\beta^2 S^2} - \frac{1}{2f\beta^2 S}\right) = 0,$$

Of the two roots of this quadratic, one is always negative and can be ignored. The other root is

$$K = \frac{1}{\beta}\left(\sqrt{\frac{1}{2fS}} - \frac{1}{S}\right).$$

In the phase diagram in Figure **??**, the dashed lines represent the zero-isoclines and the solid lines the trajectories.

Equating the two zero-isoclines and rearranging results in a fifth order polynomial in S:

$$S(1 - \beta(1 - S)S)^2 - 2f = 0,$$

implying that there could be as many as five equilibria. Expanding this polynomial, the coefficients are

$$\beta^2 S^5 - 2\beta^2 S^4 + (2\beta + \beta^2)S^3 - 2\beta S^2 + S - 2f = 0.$$

The equilibria are thus determined by the two parameters, β and f. For specific parameter values the equilibria can be found by passing these coefficient values to a standard polynomial root finding algorithm (e.g., the intrinsic Matlab function `ROOTS`).

There are 3 long-run equilibria in this system, two that are locally stable (points A and C) and one that is a saddlepoint (point B). The state space is divided into two regions of attraction, one in which the system moves toward point A and the other towards point C. The dividing line between these regions consists of points that move the system towards point B.

Notice that, due to the choice of scaling parameters, the zero-isocline for S crosses the S-axis at $S = 1$ and the K axis at $K = 1$ (see Figure **??**). The zero-isocline for K does not cross the K axis; as S approaches 0, K approaches $-\infty$. It crosses the S axis at the point $S = 2f$ and is increasing at that point. If this point is greater than 1 the two isoclines will never cross at positive stock and industry levels. A necessary and sufficient condition for the existence of an equilibrium, therefore, is that $f < \frac{1}{2}$.[6]

A restriction on β comes from examining the expression for the $S' = 0$. The isocline will exhibit a singularity (i.e., K becomes infinite) whenever $\beta(1-S)S = 1$. $(1-S)S$ is always less than $1/4$ and hence $\beta < 4$ is necessary to avoid a singularity. This information is very useful because it means that we only need to explore solutions for $\beta \in [0, 4)$ and $f \in [0, \frac{1}{2})$. Furthermore, the parameter δ does not affect the equilibria, only the speed at which the system moves toward equilibrium. In particular, higher values of δ cause faster adjustments in the industry size.

To summarize, the fishery model is an example of a system that can be characterized by a set of differential equations and initial conditions. The behavior of the system can, therefore, be studied by starting it at any specific point in the state space and propagating through time. For two (or possibly three) dimensional systems, the global behavior of the system can be usefully represented by the phase diagram. It is often useful to study the equilibria of such a system by calculating the roots of the zero-isocline curves.

Exercises

1. Derive the $O(h^2)$ backward difference approximation to $f'(x)$ at the points x, $x + h$ and $x + \lambda h$ given above.

2. Derive an $O(h)$ centered approximation to $f''(x)$ at the points $x - \lambda h$, x and $x + h$ (hint: proceed as above using the first order difference approximation to $f'(x)$) Why is this approximation only $O(h)$ and not $O(h^2)$?

3. A basic biological model for predator-prey interactions, known as the Lokta-Volterra model, can be written

$$x' = \alpha x - xy$$

[6] A curious feature of this model is that, for small S ($S < 2f$) K can become negative (this could be remedied by defining $K' = \delta K \pi$).

$$y' = xy - y,$$

where x is the population of a prey species and y is the population of a predator species. To make sense we restrict attention to $x, y > 0$ and $\alpha > 0$ (the model is scaled to eliminate excess parameters; you should determine how many scaling dimensions the model has). Although admittedly a simple model, it captures some of the essential features of the relationship. First the prey population grows at rate α when there are no predators present and the greater the number of predators, the slower the population grows and declines when the predator population exceeds α. The predator population, on the other hand declines if it grows too large unless prey is plentiful. Determine the equilibria (there are two) and draw the phase diagram [hint: this model exhibits cycles].

4. Demand for a commodity is given by $q = 2p^{-0.5}$. The price of a good falls from 4 to 1. Compute the change in consumer surplus:

   (a) analytically using Calculus;

   (b) numerically using a 10 interval trapezoid rule;

   (c) numerically using a 10 interval Simpson rule;

   (d) numerically using a 10 point Gauss-Legendre rule.

5. For $z > 0$, the cumulative probability function for a standard normal random variable is given by

$$F(z) = 0.5 + \frac{1}{\sqrt{2\pi}} \int_0^z \exp\{-\frac{x^2}{2}\}\, dx.$$

   (a) Write a short Matlab program that will estimate the value of $F(z)$ using Simpson's rule. The program should accept z and the number of intervals n in the discretization as input; the program should print $F(z)$.

   (b) What values of $F(z)$ do you obtain for $z = 1$ and $n = 6$, $n = 10$, $n = 20$ $n = 50$, $n = 100$? How do these values compare to published statistical tables?

6. Using Monte Carlo integration, estimate the expectation of $f(\tilde{X}) = 1/(1 + \tilde{X}^2)$ where $\tilde{X}$ is exponentially distributed with CDF $F(x) = 1 - \exp(-x)$ for $x \geq 0$. Compute an estimate using 100, 500, and 1000 replicates.

7. A government stabilizes the supply of a commodity at $S = 2$, but allows the price to be determined by the market. Domestic and export demand for the commodity are given by:

$$\begin{aligned} D &= \tilde{\theta}_1 P^{-1.0} \\ X &= \tilde{\theta}_2 P^{-0.5}, \end{aligned}$$

where $\log \tilde{\theta}_1$ and $\log \tilde{\theta}_2$ are normally distributed with means 0, variances 0.02 and 0.01, respectively, and covariance 0.01.

   (a) Compute the expected price Ep and the ex-ante variance of price Vp using a 6^{th} degree Gaussian discretization for the demand shocks.

   (b) Compute the expected price Ep and the ex-ante variance of price Vp using a 1000 replication Monte Carlo integration scheme.

   (c) Repeat parts (a) and (b) assuming the log of the demand shocks are negatively correlated with covariance -0.01.

8. Consider the commodity market model of Chapter 1, except now assume that log yield is normally distributed with mean 0 and standard deviation 0.2.

   (a) Compute the expectation and the variance of price without government support payments.

   (b) Compute the expectation and the variance of the effective producer price assuming a support price of 1.

9. Consider a market for an agricultural commodity in which farmers receive a government deficiency payment whenever the market price p drops below an announced target price $\bar{p}$. In this market, producers base their acreage planting decisions on their expectation of the effective producer price $f = \max(p, \bar{p})$; specifically, acreage planted a is given by:

$$a = 1 + (Ef)^{0.5}.$$

Production q is acreage planted a times a random yield $\tilde{y}$, unknown at planting time:

$$q = a \cdot \tilde{y};$$

and quantity demanded at harvest is given by

$$q = p^{-0.2} + p^{-0.5}.$$

Conditional on information known at planting time, $\log y$ is normally distributed with mean 0 and variance 0.03. For $\bar{p} = 0$, $\bar{p} = 1$, and $\bar{p} = 2$, compute:

(a) the expected subsidy $E[q(f - p)]$;

(b) the ex-ante expected producer price Ef;

(c) the ex-ante variance of producer price Vf;

(d) the ex-ante expected producer revenue $E[fq]$; and

(e) the ex-ante variance of producer revenue $V[fq]$.

10. Suppose acreage planted at the beginning of the growing season is given by $a = \alpha(Ep, Vp)$ where p is price at harvest time and E and V are the expectation and variance operators conditional on information known at planting time. Further suppose that $p = \pi(ay)$ where yield y is random and unknown at planting time. Develop an algorithm for computing the acreage planted under rational expectations.

11. Professor Jones, a well-known econometrician, argues that the best way to approximate a real-valued function with no closed-form expression over an interval is to (1) evaluate the function at n equally-spaced points and then (2) fit an m-degree polynomial to the points, using ordinary least squares to compute the coefficients on the x^i terms, $i = 0, 1, 2, \ldots, m$. To improve the approximation, he further argues, increase n until the standard errors are tolerably close to zero.

    Is Jones's approach sensible? If not, what method would you recommend? Justify your method using language that Jones is capable of understanding.

12. Professor Sayan, a regional economist, maintains a large deterministic model of the Turkish economy. Using his model, Professor Sayan can estimate the number of new jobs y that will be created under the new GATT agreement. However, Dr. Sayan is unsure about the value of one critical model parameter, the elasticity of labor supply x. A

recent econometric study estimated the elasticity to be $\bar{x}$ and gave an asymptotic normal standard error σ. Given the uncertainty about the value of x, Dr. Sayan wishes to place a confidence interval around his estimate of y. He has considered using Monte Carlo methods, drawing pseudo-random values of x according to the published distribution and computing the value of y for each x. However, a large number of replications is not feasible because two hours of mainframe computer time are needed to solve the model each time. Do you have a better suggestion for Dr. Sayan? Justify your answer.

Chapter 6

Function Approximation

In many computational economic applications, one must approximate an analytically intractable real-valued function f with a computationally tractable function $\hat{f}$.

Two types of function approximation problems arise often in computational economic applications. In the *interpolation* problem, one is given or otherwise uncovers some properties satisfied by the function f and then must choose an approximant $\hat{f}$ from a family of 'nice', tractable functions that satisfies those properties. The data available about f is often just its value at a set of specified points. The data, however, could include first or higher derivatives of f at some of the points.

Interpolation methods were originally developed to approximate the value of mathematical and statistical functions from published tables of values. In most modern computational economic applications, however, the analyst is free to chose what data to obtain about the function to be approximated. Modern interpolation theory and practice is concerned with ways to optimally extract data from a function and with computationally efficient methods for constructing and working with its approximant.

In the *functional equation* problem, one must find a function f that satisfies

$$Tf = g$$

where T is an operator that maps a vector space of functions into itself and g is a known function in that space. In the equivalent *functional fixed-point* problem, one must find a function f such that

$$Tf = f.$$

Functional equations are common in dynamic economic analysis. For example, the Bellman equation that characterizes the solutions of a dynamic optimization model is a functional fixed-point equation. The Euler equation and the fundamental asset pricing differential equation are also functional equations.

Functional equations are difficult to solve because the unknown is not simply a vector in $\Re^n$, but an entire function f whose domain contains an infinite number points. Moreover, the functional equation typically imposes an infinite number of conditions on the solution f. Except in very few special cases, functional equations lack analytic closed-form solutions and thus cannot be solved exactly. One must therefore settle for an approximate solution $\hat{f}$ that satisfies the functional equation closely. In many cases, one can compute accurate approximate solutions to functional equations using techniques that are natural extensions interpolation methods.

Numerical analysts have studied function approximation and functional equation problems extensively and have acquired substantial experience with different techniques for solving them. In this chapter we discuss methods for approximating functions and focus on the two most generally practical techniques: Chebychev polynomial and polynomial spline approximation. Univariate function interpolation methods are developed first and then are generalized to multivariate function interpolation methods. In the final section, we introduce the collocation method, a natural generalization of interpolation methods that may be used to solve a variety of functional equations.

6.1 Interpolation Principles

Interpolation is the most generally practical method for approximating a real-valued function f defined on an interval of the real line $\Re$. The first step in designing an interpolation scheme is to specify a series of n linearly independent *basis functions* $\phi_1, \phi_2, \ldots, \phi_n$ which will be used to represent the approximant. The approximant $\hat{f}$ will be written as a linear combination of the basis functions

$$\hat{f}(x) = \sum_{j=1}^{n} c_j \phi_j(x),$$

whose *basis coefficients* $c_1, c_2, \ldots, c_n$ are to be determined. Polynomials of increasing order are often used as basis functions, although other types of

basis functions, most notably spline functions, are also commonly used.[1] The number n of independent basis functions is called the *degree of interpolation*.

The second step in designing an interpolation scheme is to specify the properties of the original function f that one wishes the approximant $\hat{f}$ to replicate. Because there are n undetermined coefficients, n conditions are required to fix the approximant. The easiest and most common conditions imposed are that the approximant *interpolate* or match the value of the original function at selected *interpolation nodes* $x_1, x_2, \ldots, x_n$.

Given n interpolation nodes and n basis functions, computing the basis coefficients reduces to solving a linear equation. Specifically, one fixes the n undetermined coefficients $c_1, c_2, \ldots, c_n$ of the approximant $\hat{f}$ by solving the *interpolation conditions*

$$\sum_{j=1}^{n} c_j \phi_j(x_i) = f(x_i) = y_i \qquad \forall i = 1, 2, \ldots, n.$$

Using matrix notation, the interpolation conditions equivalently may be written as the matrix linear *interpolation equation* whose unknown is the vector of basis coefficients c:

$$\Phi c = y.$$

Here,

$$\Phi_{ij} = \phi_j(x_i)$$

is the typical element of the *interpolation matrix* Φ. In theory, an interpolation scheme is well-defined if the interpolation nodes and basis functions are chosen such that the interpolation matrix is nonsingular.

Interpolation schemes are not limited to using only function value information. In many applications, one may wish to interpolate both function values and derivatives at specified points. This would be the case, for example, if solving an initial value problem, which was discussed in the preceding chapter. Suppose, for example, that one wishes to construct an approximant $\hat{f}$ that replicates the function's values at nodes $x_1, x_2, \ldots, x_{n_1}$ and its first derivatives at nodes $x'_1, x'_2, \ldots, x'_{n_2}$. An approximant that satisfies these

[1] Approximations that are non-linear in basis function exist (e.g. rational approximations), but are more difficult to work with and hence are not often seen in practical applications.

conditions may be constructed by selecting $n = n_1 + n_2$ basis functions and fixing the basis coefficients $c_1, c_2, \ldots, c_n$ of the approximant by solving the interpolation equation

$$\begin{aligned} \sum_{j=1}^{n} c_j \phi_j(x_i) &= f(x_i), \qquad \forall i = 1, \ldots, n_1 \\ \sum_{j=1}^{n} c_j \phi_j'(x_i') &= f'(x_i'), \qquad \forall i = 1, \ldots, n_2 \end{aligned}$$

for the undetermined coefficients c_j. This principle applies to any combination of function values, derivatives, or even antiderivatives at selected points. All that is required is that the associated interpolation matrix be nonsingular.

In developing an interpolation scheme, the analyst should chose interpolation nodes and basis functions that satisfy certain criteria. First, the approximant should be capable of producing an accurate approximation for the original function f. In particular, the interpolation scheme should allow the analyst to achieve, at least in theory, an arbitrarily accurate approximation by increasing the degree of approximation. Second, it should be possible to compute the basis coefficients quickly and accurately. In particular, the interpolation equation should be well-conditioned and should be easy to solve—diagonal, near diagonal, or orthogonal interpolation matrices are best. Third, the approximant should be easy to work with. In particular, the basis functions should be easy and relatively costless to evaluate, differentiate, and integrate.

Interpolation schemes may be classified as either *spectral* methods or *finite element* methods. A spectral method uses basis functions that are nonzero over the entire domain of the function being approximated, except possibly at a finite number of points. In contrast, a finite element method uses basis functions that are nonzero over only a subinterval of the domain of approximation. Polynomial interpolation, which uses polynomials of increasing degree as basis functions, is the most common spectral method. Spline interpolation, which uses basis functions that are polynomials of small degree over subintervals of the approximation domain, is the most common finite element method. We examine both of these methods in greater detail in the following sections.

6.2 Polynomial Interpolation

According to the Weierstrass Theorem, any continuous real-valued function f defined on a bounded interval $[a, b]$ of the real line can be approximated to any degree of accuracy using a polynomial. More specifically, if $\epsilon > 0$, there exists a polynomial p such that

$$||f - p|| = \sup_{x \in [a,b]} |f(x) - p(x)| < \epsilon.$$

The Weierstrass theorem provides strong motivation for using polynomials to approximate continuous functions. The theorem, however, is not very practical. It gives no guidance on how to find a good polynomial approximant. It does not even state what order polynomial is required to achieve the required level of accuracy.

One apparently reasonable way to construct a n^{th}-degree polynomial approximant for a function f is to form the unique $(n-1)^{th}$-order polynomial

$$p(x) = c_1 + c_2 x + c_3 x^2 + \ldots + c_n x^{n-1}$$

that interpolates f at the n evenly spaced interpolation nodes

$$x_i = a + \frac{i-1}{n-1}\Big(b - a\Big) \qquad \forall i = 1, 2, \ldots, n.$$

In practice, however, polynomial interpolation at evenly spaced nodes often does not produce an accurate approximant. In fact, there are well-behaved functions for which evenly spaced node polynomial approximants rapidly deteriorate, rather than improve, as the degree of approximation n rises.

Numerical analysis theory and empirical experience both suggest that polynomial approximants over a bounded interval $[a, b]$ should be constructed by interpolating the underlying function at the so-called *Chebychev nodes:*

$$x_i = \frac{a+b}{2} + \frac{b-a}{2} \cos\left(\frac{n - i + 0.5}{n}\pi\right), \qquad \forall i = 1, 2, \ldots, n.$$

As illustrated in Figure 6.1, the Chebychev nodes are not evenly spaced. They are more closely spaced near the endpoints of the interpolation interval and less so near the center.

Chebychev-node polynomial interpolants possess some strong theoretical properties. According to Rivlin's Theorem, Chebychev-node polynomial interpolants are very nearly optimal polynomial approximants. Specifically, the

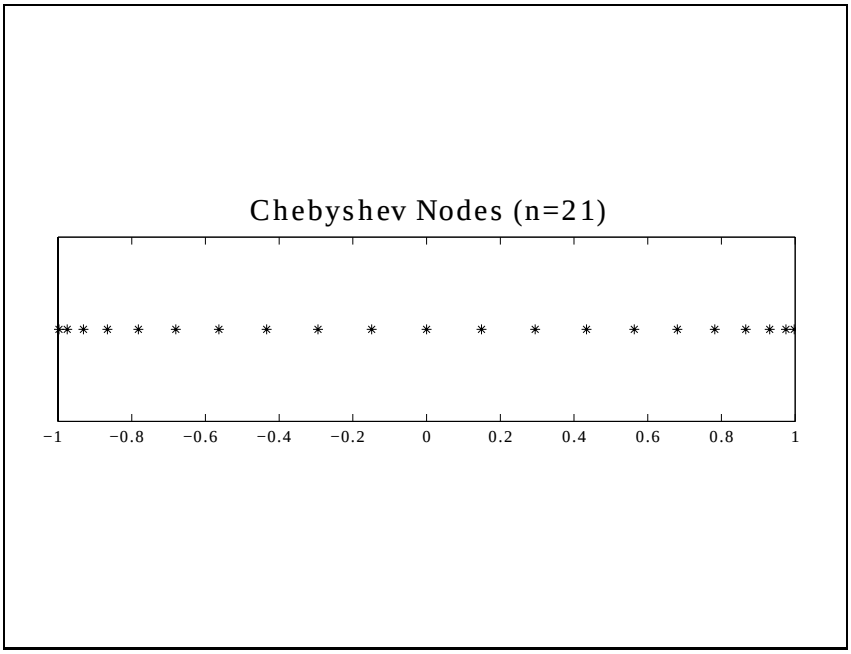

Figure 6.1

approximation error associated with the n^{th}-degree Chebychev-node polynomial interpolant cannot larger than $2\pi \log(n)+2$ times the lowest error attainable with any other polynomial approximant of the same order. For $n = 100$, this factor is approximately 30, which is very small when one considers that other polynomial interpolation schemes typically produce approximants with errors that are orders of magnitude, that is, powers of 10, larger then the optimum. In practice, the accuracy afforded by the Chebychev-node polynomial interpolant is often much better than indicated by Rivlin's bound, especially if the function being approximated is smooth.

Another theorem, Jackson's theorem, implies a more useful result. Specifically, if f is continuously differentiable, then the approximation error afforded by the n^{th}-degree Chebychev-node polynomial interpolant p_n can be bounded above:

$$||f - p_n|| \leq \frac{6}{n}||f'||(b-a)(\log(n)/\pi + 1).$$

This error bound can often be accurately estimated in practice, giving the analyst a good indication of the accuracy afforded by the Chebychev-node

polynomial interpolant. More importantly, however, the error bound goes to zero as n rises. That is, unlike for evenly spaced node polynomial interpolation, one can achieve any desired degree of accuracy with Chebychev-node polynomial interpolation by increasing the degree of approximation.

To illustrate the difference between Chebychev and evenly spaced node polynomial interpolation, consider approximating the function $f(x) = \exp(-x)$ on the interval $[-1, 1]$. The approximation error associated with ten node polynomial interpolants are illustrated in Figure 6.2. The Chebychev node polynomial interpolant exhibits errors that oscillate fairly evenly throughout the interval of approximation, a common feature of Chebychev node interpolants. The evenly spaced node polynomial interpolant, on the other hand, exhibits significant instability near the endpoints of the interval. The Chebychev node polynomial interpolant avoids endpoint instabilities because the nodes are more heavily concentrated near the endpoints.

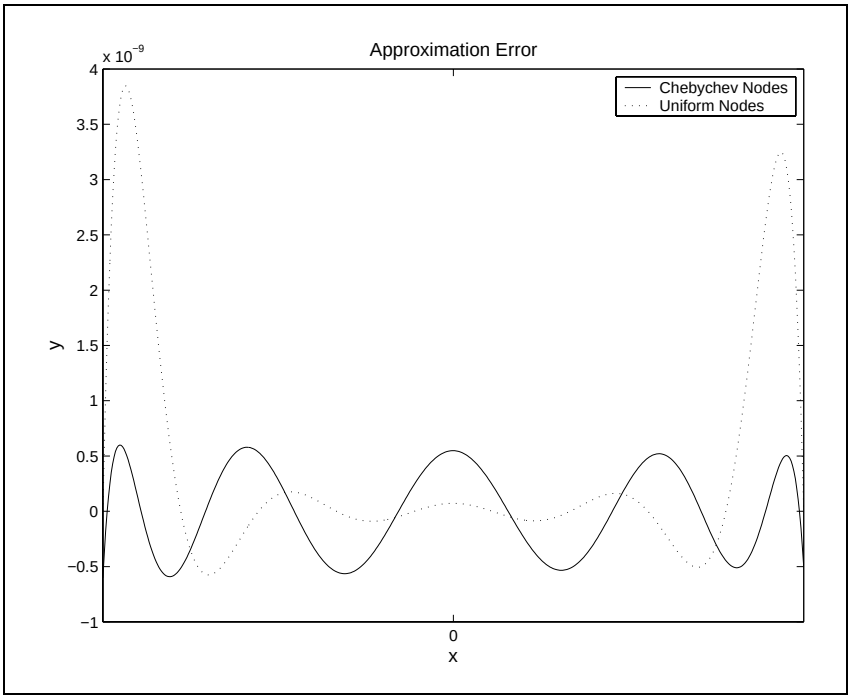

Figure 6.2

The most intuitive basis for expressing polynomials, regardless of the interpolation nodes chosen, is the *monomial basis* consisting of the simple power

functions $1, x, x^2, x^3, \ldots$, illustrated in Figure 6.3. However, the monomial basis produces an interpolation matrix Φ that is a so-called Vandermonde matrix:

$$\Phi = \begin{bmatrix} 1 & x_1 & \ldots & x_1^{n-2} & x_1^{n-1} \\ 1 & x_2 & \ldots & x_2^{n-2} & x_2^{n-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & x_n & \ldots & x_n^{n-2} & x_n^{n-1} \end{bmatrix}.$$

Vandermonde matrices are notoriously ill-conditioned, and increasingly so as the degree of approximation n is increased. Thus, efforts to compute the basis coefficients of the monomial basis polynomials often fail due to rounding error, and attempts to compute increasingly more accurate approximations by raising the number of interpolation nodes are often futile.

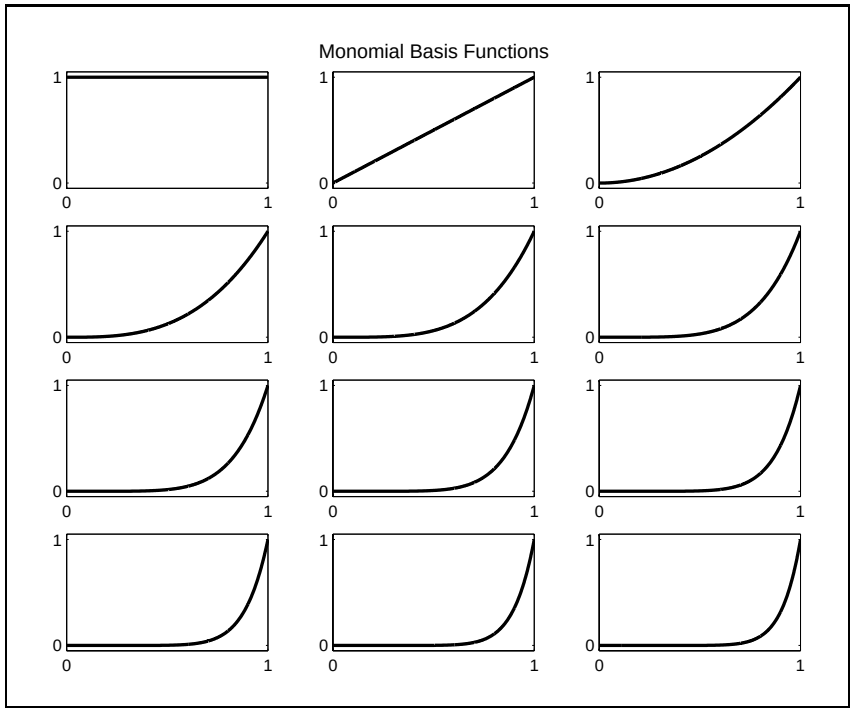

Figure 6.3

Fortunately, alternatives to the standard monomial basis exist. In fact, any sequence of n polynomials having exact orders $0, 1, 2, \ldots, n-1$ can serve

as a basis for all polynomials of order less than n. One such basis for the interval $[a, b]$ on the real line is the Chebychev polynomial basis. The Chebychev polynomials are defined recursively as:[2]

$$\phi_j(x) = T_{j-1}\left(2\frac{x-a}{b-a} - 1\right)$$

where, for $z \in [-1, 1]$,

$$\begin{aligned} T_0(z) &= 1 \\ T_1(z) &= z \\ T_2(z) &= 2z^2 - 1 \\ T_3(z) &= 4z^3 - 3z \\ &\vdots \\ T_j(z) &= 2zT_{j-1}(z) - T_{j-2}(z). \end{aligned}$$

The first twelve Chebychev basis polynomials for the interval $[0, 1]$ are displayed in Figure 6.4.

Chebychev polynomials are an excellent basis for constructing polynomials that interpolate function values at the Chebychev nodes. Chebychev basis polynomials in combination with Chebychev interpolation nodes yields an extremely well-conditioned interpolation equation that can be accurately and efficiently solved, even for high degrees of interpolation. The interpolation matrix Φ associated with the Chebychev interpolation has typical element

$$\Phi_{ij} = \cos((n - i + 0.5)(j - 1)\pi/n).$$

This Chebychev interpolation matrix is orthogonal

$$\Phi'\Phi = \text{diag}\{n, n/2, n/2, \ldots, n/2\}$$

and has a condition number $\sqrt{2}$ regardless of the degree of interpolation, which is very near the ideal minimum of 1. This implies that the Chebychev basis coefficients can be computed quickly and accurately, regardless of the degree of interpolation.

[2]The Chebychev polynomials also possess the alternate trigonometric definition $T_j(z) = \cos\left(\arccos(z)j\right)$ on the domain $[a, b]$.

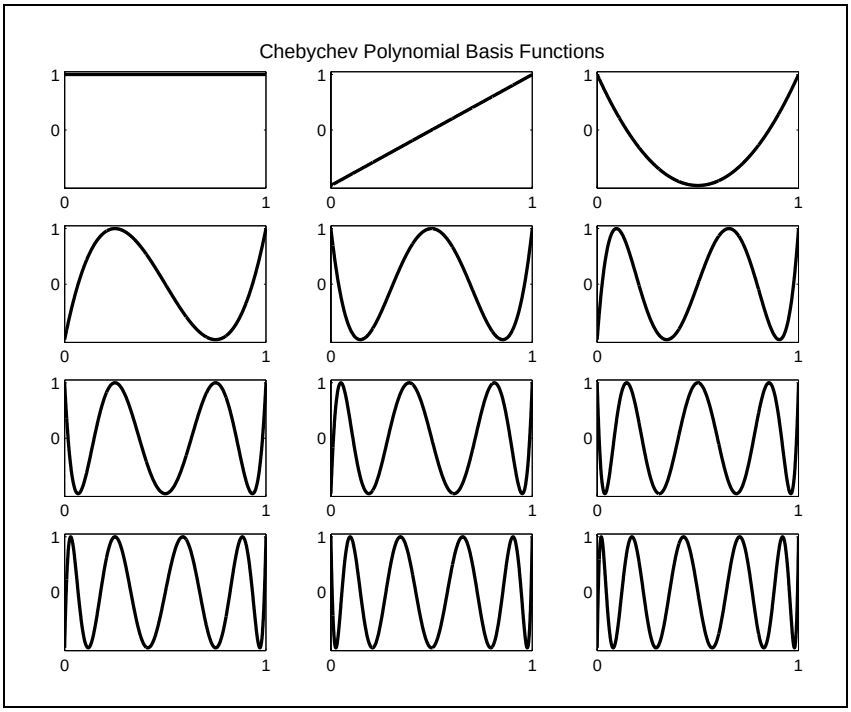

Figure 6.4

6.3 Piecewise Polynomial Splines

Piecewise polynomial splines, or simply splines for short, are a rich, flexible class of functions that may be used instead of high degree polynomials to approximate a real-valued function over a bounded interval. Generally, a k^{th} order spline consists of series of k-degree polynomial segments spliced together so as to preserve continuity of derivatives of order $k-1$ or less. The points at which the polynomial pieces are spliced together, $\nu_1 < \nu_2 < \ldots < \nu_p$, are called the *breakpoints* of the spline. By convention, the first and last breakpoints are the endpoints of the interval of approximation $[a, b]$.

A general order k spline with p breakpoints may be characterized by $(p-1)(k+1)$ parameters, given that each of the $p-1$ polynomial segments is defined by its $k+1$ coefficients. By definition, however, a spline is required to be continuous and have continuous derivatives up to order $k-1$ at each of the $p-2$ interior breakpoints, which imposes $k(p-2)$ conditions. Thus,

a k order spline with p breakpoints is actually characterized by $n = (k+1)(p-1) - k(p-2) = p + k - 1$ free parameters. It should not be surprising that a general k order spline with p breakpoints can be written as a linear combination of $n = p + k - 1$ basis functions.

There are many ways to express bases for splines, but for applied numerical work the most useful are the so-called B-splines. The B-splines for an order k splines with breakpoint vector ν can be computed using the recursive definition

$$B_j^{k,\nu}(x) = \frac{x - \nu_{j-k}}{\nu_j - \nu_{j-k}} B_{j-1}^{k-1,\nu}(x) + \frac{\nu_{j+1} - x}{\nu_{j+1} - \nu_{j+1-k}} B_j^{k-1,\nu}(x),$$

for $i = 1, \ldots, n$, with the recursion starting with

$$B_j^{0,\nu}(x) = \begin{cases} 1 & \text{if } \nu_j \leq x < \nu_{j+1} \\ 0 & \text{otherwise} \end{cases} .$$

This definition requires that we extend the breakpoint vector, ν, for $j < 1$ and $j > p$:

$$\nu_j = \begin{cases} a & \text{if } j \leq 1 \\ b & \text{if } j \geq p \end{cases} .$$

Additionally, at the endpoints we set the terms

$$\frac{B_0^{k-1,\nu}}{\nu_1 - \nu_{1-k}} = \frac{B_n^{k-1,\nu}}{\nu_{n+1} - \nu_{n-k+1}} = 0.$$

Given a B-spline representation of a spline, the spline can easily be differentiated by computing simple differences, and can be integrated by computing simple sums. Specifically:

$$\frac{dB_j^{k,\nu}(x)}{dx} = \frac{k}{\nu_j - \nu_{j-k}} B_{j-1}^{k-1,\nu}(x) - \frac{k}{\nu_{j+1} - \nu_{j+1-k}} B_j^{k-1,\nu}(x)$$

and

$$\int_a^x B_j^{k,\nu}(z)dz = \sum_{i=j}^{n} \frac{\nu_i - \nu_{i-k}}{k} B_{i+1}^{k+1,\nu}(x).$$

Although these formulae appear a bit complicated, their application in computer programs is relatively straightforward. First notice that the derivative of a B-spline of order k is a weighted sum of two order $k-1$ B-splines. Thus,

the derivative of an order k spline is an order $k-1$ spline with the same breakpoints. Similarly, the integral of a B-spline can be represented in terms of two B-splines of order $k+1$ splines. Thus, the antiderivative of an order k spline is an order $k+1$ spline with the same breakpoints.

Two classes of splines are often employed in practice. A first-order or *linear spline* is a series of line segments spliced together to form a continuous function. A third-order or *cubic spline* is a series of cubic polynomials segments spliced together to form a twice continuously differentiable function.

Linear spline approximants are particularly easy to construct and evaluate in practice, which explains their widespread popularity. Linear splines use line segments to connect points on the graph of the function to be approximated. A linear spline with n evenly spaced breakpoints on the interval $[a, b]$ may be written as a linear combination

$$\hat{f}(x) = \sum_{i=1}^{n} c_i \phi_i(x)$$

of the basis functions:

$$\phi_j(x) = \begin{cases} 1 - \frac{|x-\nu_j|}{w} & \text{if } |x-\nu_j| \leq w \\ 0 & \text{otherwise} \end{cases}$$

Here, $w = (b-a)/(n-1)$ is the distance between breakpoints and $\nu_j = a+(j-1)w$, $j = 1, 2, \ldots, n$, are the breakpoints. The linear spline basis functions are popularly called the "hat" functions, for reasons that are clear from Figure 6.5. This figure illustrates the basis function for twelve-degree, evenly spaced breakpoint linear splines on the interval $[0, 1]$. Each hat function is zero everywhere, except over a narrow support element of width $2w$. The basis function achieves a maximum of 1 at the midpoint of its support element.

One can fix the coefficients of an n-degree linear spline approximant for a function f by interpolating its values at any n points of its domain, provided that the resulting interpolation matrix is nonsingular. However, if the interpolation nodes $x_1, x_2, \ldots, x_n$ are chosen to coincide with the spline breakpoints $\nu_1, \nu_2, \ldots, \nu_n$, then computing the basis coefficients of the linear spline approximant becomes a trivial matter. If the interpolation nodes and breakpoints coincide, then $\phi_i(x_j)$ equals one if $i = j$, but equals zero otherwise. That is, the interpolation matrix Φ is simply the identity matrix and the interpolation equation reduces to the trivial identity $c = y$ where y is

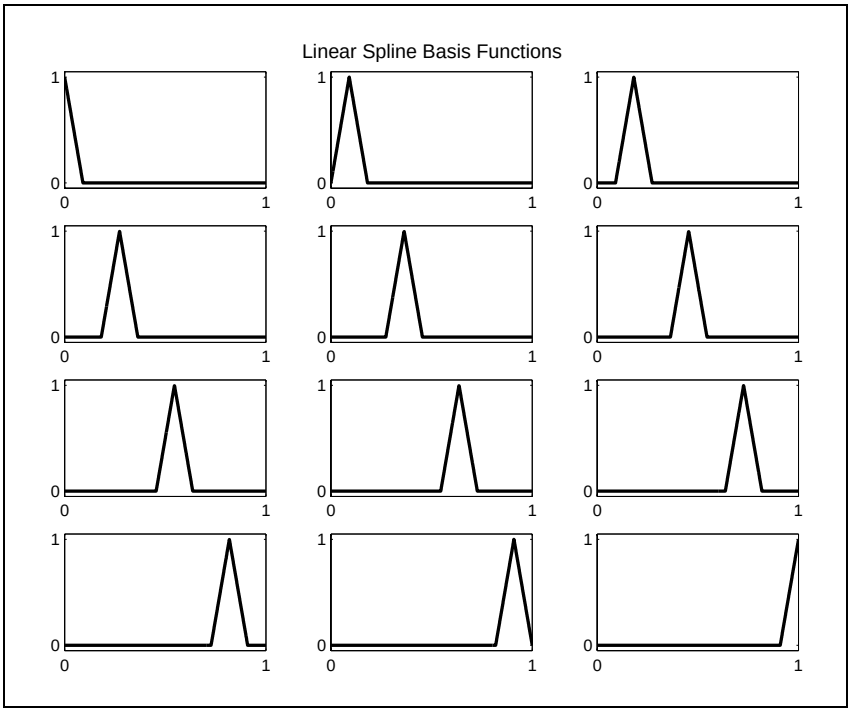

Figure 6.5

the vector of function values at the interpolation nodes. The linear spline approximant of f when nodes and breakpoints coincide thus takes the form

$$\hat{f}(x) = \sum_{i=1}^{n} f(x_i)\phi_i(x).$$

When interpolation nodes and breakpoints coincide, no computations other than function evaluations are required to form the linear spline approximant. For this reason linear spline interpolation nodes in practice are always chosen to be the spline's breakpoints.

Evaluating a linear spline approximant and its derivative at an arbitrary point x is also straightforward. Since at most two basis functions are nonzero at any point, only two basis function evaluations are required. Specifically, if i is the greatest integer less than $1 + (x - a)/w$, then x lies in the interval $[x_i, x_{i+1}]$. Thus,

$$\hat{f}(x) = (c_{i+1}(x - x_i) + c_i(x_{i+1} - x))/w$$

and

$$\hat{f}'(x) = (c_{i+1} - c_i)/w.$$

Higher order derivatives are zero, except at the breakpoints, where they are undefined.

Linear splines are attractive for their simplicity, but have certain limitations that often make them a poor choice for computational economic applications. By construction, linear splines produce first derivatives that are discontinuous step functions and second derivative that are zero almost everywhere. Linear spline approximants thus typically do a very poor job of approximating the first derivative of a nonlinear function and are incapable of approximating its second derivative. In some economic applications, the derivative represents a measure of marginality that is of as much interest to the analyst as the function itself. In other applications, the first and maybe second derivative of the function may be needed to solve for the root of the function using Newton-like method.

Cubic spline approximants offer a higher degree of smoothness while retaining much of the flexibility and simplicity of linear spline approximants. Because cubic splines possess continuous first and second derivatives, they typically produce adequate approximations for both the function and its first and second derivatives.

The basis functions for n-degree, evenly spaced breakpoint cubic splines on the interval $[a, b]$ are generated using the $n-2$ breakpoints $\nu_j = a+w(j-1)$, $j = 1, 2, \ldots, n-2$, where $w = \frac{b-a}{n-3}$. Cubic spline basis function generated with evenly spaced breakpoints are nonzero over a support element of width $4w$. As such, at any point of $[a, b]$, at most four basis functions are nonzero. The basis functions for twelve-degree, evenly spaced breakpoint cubic splines on the interval $[0, 1]$ are illustrated in Figure 6.6.

Although spline breakpoints are often chosen to be evenly spaced in most applications, this need not be the case. Indeed, the ability to distribute breakpoints unevenly and to stack them on top of one another adds considerably to the flexibility of splines, allowing them to accurately approximate a wide range of functions. In general, functions that exhibit wide variations in curvature are difficult to approximate numerically with entire polynomials of high degree. With splines, however, one can often finesse curvature difficulties by concentrating breakpoints in regions displaying the highest degree of curvature.

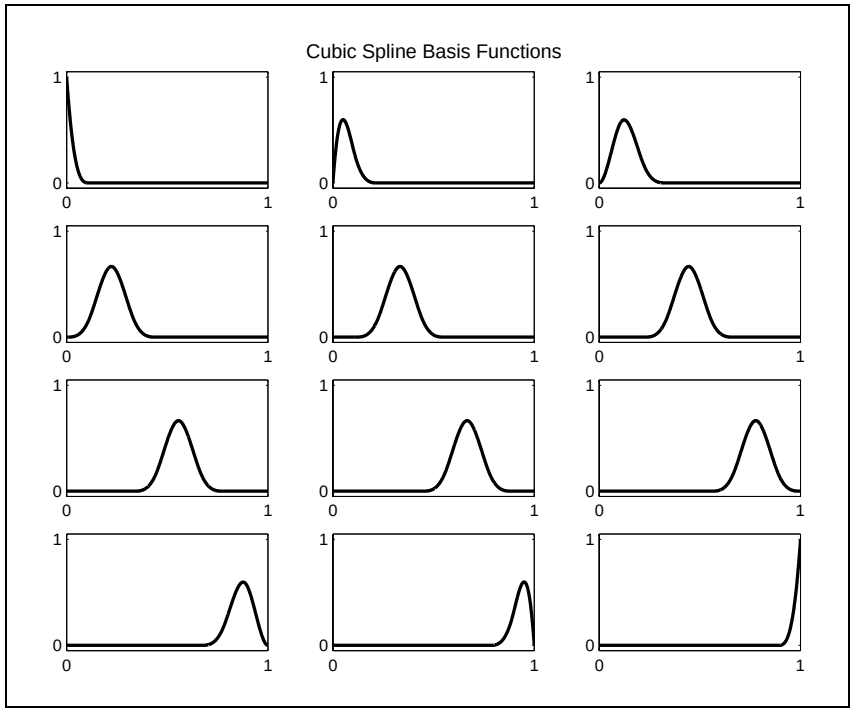

Figure 6.6

To illustrate the importance of breakpoint location, consider the problem of forming a cubic spline approximant for Runge's function

$$f(x) = \frac{1}{1 + 25x^2} \text{ for } x \in [-5, 5].$$

Figure 6.7 displays two cubic spline approximations, one using thirteen evenly spaced breakpoints, the other using thirteen breakpoints that cluster around zero (the breakpoints are indicated by 'x' symbols). Figure 6.8 shows the associated approximation errors (note that the errors for the unevenly spaced approximation have been multiplied by 100). In Figure 6.7 the unevenly spaced breakpoints approximation lies almost on top of the actual function, whereas the even spacing leads to significant errors, especially near zero. The figures clearly demonstrates the power of spline approximations with good breakpoint placement.

The placement of the breakpoints can also be used to affect the continuity of the spline approximant and its derivatives. By stacking breakpoints on top

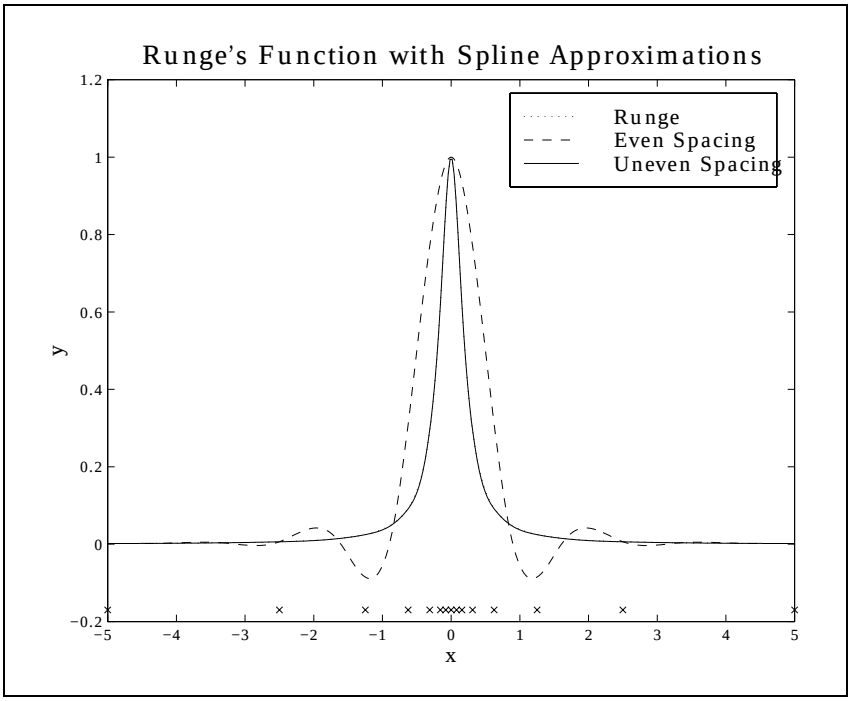

Figure 6.7

of one another, we can reduce the smoothness at the breakpoints. Normally, an order k spline has continuous derivatives to order $k-1$ at the breakpoints. By stacking q breakpoints, we can reduce this to $k-q$ continuous derivatives at this breakpoint. For example, with two equal breakpoints, a cubic spline possesses a discontinuous second derivative at the point. With three equal breakpoints, a cubic spline possesses a discontinuous first derivative at that point, that is, it exhibits a kink there. Stacking breakpoints is a useful practice if the function is known a priori to exhibit a kink at a given point, a not uncommon occurrence in practice.

Regardless of the placement of breakpoints, splines have several important and useful properties. We have already commented on the limited domain of the basis function. This limited support implies that spline interpolation matrices are sparse and for this reason can be stored and manipulated as sparse matrices. This property is extremely useful in high-dimensional problems for which a fully expanded interpolation matrix would strain any computer's memory. Another useful feature of splines is that their values

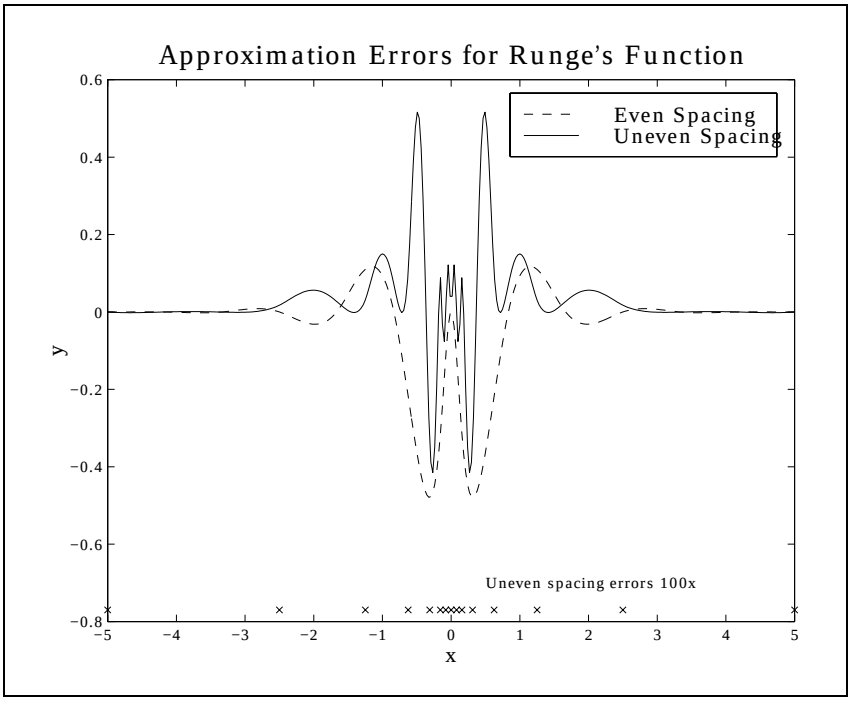

Figure 6.8

are bounded, thereby reducing the likelihood that scaling effects will cause numerical difficulties. In general, the limited support and bounded values make spline basis matrices well-conditioned.

If the spline interpolation matrix must be reused, one must resist the temptation to form and store its inverse, particularly if the size of the matrix is large. Inversion destroys the sparsity structure. More specifically, the inverse of the interpolation matrix will be dense, even though the interpolation matrix is not. When n is large, solving the sparse n by n linear equation using sparse L-U factorization will generally be less costly than performing the matrix-vector multiplication required with the dense inverse interpolation matrix.

6.4 Multidimensional Interpolation

The univariate interpolation methods discussed in the preceding sections may be extended in a natural way to multivariate functions through the use of tensor products. To illustrate, consider the problem of approximating a bivariate real-valued function $f(x, y)$ defined on a bounded interval $I = \{(x, y) \mid a_x \leq x \leq b_x, a_y \leq y \leq b_y\}$ in $\Re^2$. Suppose that ϕ_i^x, $i = 1, 2, \ldots, n_x$ and ϕ_j^y, $j = 1, 2, \ldots, n_y$ are basis functions for univariate functions defined on $[a_x, b_x]$ and $[a_y, b_y]$, respectively. Then an $n = n_x n_y$ degree basis for f on I may be constructed by letting

$$\phi_{ij}(x, y) = \phi_i^x(x)\phi_j^y(y) \qquad \forall i = 1, \ldots, n_x; j = 1, \ldots, n_y.$$

Similarly, a grid of $n = n_x n_y$ interpolation nodes can be constructed by taking the Cartesian product of univariate interpolation nodes. More specifically, if $x_1, x_2, \ldots x_{n_x}$ and $y_1, y_2, \ldots, y_{n_y}$ are n_x and n_y interpolation nodes in $[a_x, b_x]$ and $[a_y, b_y]$, respectively, then n nodes for interpolating f on I may be constructed by letting

$$\{(x_i, y_j) \mid i = 1, 2, \ldots, n_x; j = 1, 2, \ldots, n_y\}.$$

For example, suppose one wishes to approximate a function using a cubic polynomial in the x direction and a quadratic polynomial in the y direction. A tensor product basis constructed from the simple monomial basis of x and y comprises the following functions

$$1, \quad x, \quad y, \quad xy, \quad x^2, \quad y^2, \quad xy^2, \quad x^2y, \quad x^2y^2, \quad x^3, \quad x^3y, \quad x^3y^2.$$

The dimension of the basis is 12. An approximant expressed in terms of the tensor product basis would take the form

$$\hat{f}(x, y) = \sum_{i=1}^{4} \sum_{j=1}^{3} c_{ij} x^{i-1} y^{j-1}.$$

Typically, tensor product node-basis schemes inherit the favorable qualities of their univariate node-basis parents. For example, if a bivariate linear spline basis is used and the interpolation nodes $\{x_i, y_j\}$ are chosen such that the x_i and y_j coincide with the breakpoints in the x and y direction, respectively, then the interpolation matrix will be the identity matrix, just like in the univariate case. Also, if a bivariate Chebychev polynomial basis is used,

and the interpolation nodes $\{x_i, y_j\}$ are chosen such that the x_i and y_j coincide with the Chebychev nodes on $[a_x, b_x]$ and $[a_y, b_y]$, respectively, then the interpolation matrix will be orthogonal.

Tensor product schemes can be developed similarly for higher than two dimensions. Consider the problem of interpolating a d-variate function

$$f(x_1, x_2, \ldots, x_d)$$

on a d-dimensional interval

$$I = \{(x_1, x_2, \ldots, x_d) \mid a_i \leq x_i \leq b_i, i = 1, 2, \ldots, d\}.$$

If ϕ_{ij}, $j = 1, ..., n_i$ is a n_i degree univariate basis for real-valued functions of on $[a_i, b_i]$, then an approximant for f in the tensor product basis would take the following form:

$$\hat{f}(x_1, x_2, \ldots, x_d) = \sum_{j_1=1}^{n_1} \sum_{j_2=1}^{n_2} \ldots \sum_{j_d=1}^{n_d} c_{j_1 \ldots j_d} \phi_{1 j_1}(x_1) \phi_{2 j_1}(x_2) \ldots \phi_{d j_1}(x_d).$$

Using tensor notation the approximating function an be written

$$\hat{f}(x_1, x_2, \ldots, x_d) = [\phi_1(x_1) \otimes \phi_2(x_2) \otimes \ldots \otimes \phi_d(x_d)]c.$$

where c is a column vector with $n = \Pi_{i=1}^{d} n_i$ elements. An even more compact notation is

$$f(x) = \Phi(x)c$$

where $\Phi(x)$ is a function of d variables that produces an n-column row vector.

Consider the case in which $d = 2$, with $n_1 = 3$ and $n_2 = 2$, and the simple monomial (power) function bases are used (of course one should use Chebychev but it makes the example harder to follow). The elementary basis functions are

$$\begin{aligned} \phi_{11}(x_1) &= 1 \\ \phi_{21}(x_1) &= x_1 \\ \phi_{31}(x_1) &= x_1^2 \\ \phi_{12}(x_2) &= 1 \end{aligned}$$

and

$$\phi_{22}(x_2) = x_2.$$

The elementary basis vectors are

$$\begin{aligned} \phi_1(x_1) &= [1 \; x_1 \; x_1^2] \\ \text{and} & \\ \phi_2(x_2) &= [1 \; x_2]. \end{aligned}$$

Finally, the full 2–d basis vector is

$$\Phi(x) = [1 \; x_1 \; x_1^2] \otimes [1 \; x_2] = [1 \; x_2 \; x_1 \; x_1 x_2 \; x_1^2 \; x_1^2 x_2],$$

which has $n = n_1 n_2 = 6$ columns.

We are often interested in evaluating $f(x)$ at many values of x. Suppose we have an $m \times d$ matrix X, each row of which represents a single value of x, and which is denoted X_i. The matrix $\Phi(X)$ is an $m \times N$ matrix, each row of which is composed of $\phi(X_i)$ and we can write

$$f(X) = \Phi(X)c$$

to be the values of the function evaluated at each of the X_i.

Continuing the previous example, suppose we want to evaluate f at the m points [0 0], [0 0.5], [0.5 0] and [1 1]. The matrix X is thus

$$X = \begin{bmatrix} 0 & 0 \\ 0 & 0.5 \\ 0.5 & 0 \\ 1 & 1 \end{bmatrix}.$$

Then

$$\Phi(X) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0.5 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0.5 & 0 & 0.25 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix},$$

which is 4×6 ($m \times N$).

To implement interpolation in multiple dimensions it is necessary to evaluate solve the interpolation equation. If Φ_i is the degree n_i interpolation matrix associated with variable x_i, then the interpolation conditions for the multivariate function can be written

$$[\Phi_1 \otimes \Phi_2 \otimes \ldots \otimes \Phi_d]c = f(x)$$

where $f(x)$ is an n by 1 vector of function values evaluated at the interpolation nodes x, properly stacked. Using a standard result from tensor matrix algebra, the this system may be solved by forming the inverse of the interpolation matrix and postmultiplying it by the data vector:

$$c = [\Phi_1^{-1} \otimes \Phi_2^{-1} \otimes \ldots \otimes \Phi_d^{-1}] f(x);$$

Hence there is no need to invert an n by n multivariate interpolation matrix to determine the interpolating coefficients. Instead, each of the univariate interpolation matrices may be inverted individually and then multiplied together. This leads to substantial savings in storage and computational effort. For example, if the problem is 3-dimensional and the are 10 evaluation points in each dimension, only three 10 by 10 matrices need to be inverted, rather than a single 1000 by 1000 matrix.

Interpolation using tensor product schemes tends to become computationally more challenging as the dimensions rise. With a one–dimensional argument the number of interpolation nodes and the dimension of the interpolation matrix can generally be kept small with good results. For a relatively smooth function, Chebychev polynomial approximants of order 10 or less can often provide extremely accurate approximations to a function and its derivatives. If the function's argument is d-dimensional one could approximate the function using the same number of points in each dimension, but this increases the number of interpolation nodes to 10^d and the size of the interpolation matrix to 10^{2d} elements. The tendency of computational effort to grow exponentially with the dimension of the function being interpolated is known as the *curse of dimensionality*. In order to mitigate the effects of the curse requires that careful attention be paid to both storage and computational efficiency when designing and implementing numerical routines that perform approximation.

6.5 Choosing an Approximation Method

The most significant difference between spline and polynomial interpolation methods is that spline basis functions have narrow supports, but polynomial basis functions have supports that cover the entire interpolation interval. This can lead to big differences in the quality of approximation when the function being approximated is irregular. Discontinuities in the first or second derivatives can create problems for all interpolation schemes. However,

spline functions, due to their narrow support, can often contain the effects of such discontinuities. Polynomial approximants, on the other hand, allow the ill effects of discontinuities to propagate over the entire interval of interpolation. Thus, when a function exhibits kinks, spline interpolation may be preferable to polynomial interpolation.

In order to illustrate the differences between spline and polynomial interpolation, we compare in Table 6.1 the approximation error for four different functions, all defined on $[-5, 5]$, and four different approximation schemes: linear spline interpolation, cubic spline interpolation, evenly spaced node polynomial interpolation, and Chebychev polynomial interpolation. The errors are measured as the maximum absolute error using 1001 evenly spaced evaluation points on $[-5, 5]$.

The four functions are ordered in increasing difficulty of approximation. The first is polynomial and can be fit exactly by both cubic spline and polynomials "approximations". The second function is quite smooth and hence can be fit well with a polynomial. The third function (Runge's function) has continuous derivatives of all orders but has a high degree of curvature near the origin. A scaleless measure of curvature familiar to economists is $-f''/f'$; for Runge's function this measure is $1/x - 2$ which becomes unbounded at the origin. The fourth function is kinked at the origin, i.e., its derivative is not continuous.

The results presented in Table 6.1 lend support to certain rules of thumb. When comparing interpolation schemes of the same degree of approximation:

1. Chebychev node polynomial interpolation dominates evenly spaced node polynomial interpolation.

2. Cubic spline interpolation dominates linear spline interpolation, except where the approximant exhibits a profound discontinuity.

3. Chebychev polynomial interpolation dominates cubic spline interpolation if the approximant is smooth and monotonic; otherwise, cubic or even linear spline interpolation may be preferred.

Table 6.1: Errors for Selected Interpolation Methods

Function	Degree	Linear Spline	Cubic Spline	Uniform Polynomial	Chebychev Polynomial
$1 + x$	10	1.30e+001	1.71e-013	2.27e-013	1.71e-013
$+2x^2 - 3x^3$	20	3.09e+000	1.71e-013	3.53e-011	1.99e-013
	30	1.35e+000	1.71e-013	6.56e-008	3.41e-013
$\exp(-x)$	10	1.36e+001	3.57e-001	8.10e-002	1.41e-002
	20	3.98e+000	2.31e-002	2.04e-008	1.27e-010
	30	1.86e+000	5.11e-003	1.24e-008	9.23e-014
$(1 + 25x^2)^{-1}$	10	8.85e-001	9.15e-001	8.65e-001	9.25e-001
	20	6.34e-001	6.32e-001	2.75e+001	7.48e-001
	30	4.26e-001	3.80e-001	1.16e+004	5.52e-001
$\|x\|^{0.5}$	10	7.45e-001	7.40e-001	6.49e-001	7.57e-001
	20	5.13e-001	4.75e-001	1.74e+001	5.33e-001
	30	4.15e-001	3.77e-001	4.34e+003	4.35e-001

6.6 An Approximation Toolkit

Implementing routines for multivariate function approximation involves a number of bookkeeping details that are tedious at best. In this section we describe a set of numerical tools that take much of the pain out of this process. This toolbox contains several high-level functions that use a structured variable to store the essential information that defines the function space from which approximants are drawn. The toolbox also contains a set of middle-level routines that define the basis functions for Chebychev polynomials and for splines and a set of low-level utilities to handle basic computations, including tensor product manipulations. Below, all of the routines are implemented in Matlab, though in principle they can also be implemented in other computer languages.

The six high-level procedures, all prefaced by `FUN`, are `FUNDEFN`, `FUNFITF`, `FUNFITXY`, `FUNEVAL`, `FUNNODE`, and `FUNBAS`.

The most basic of these routines is `FUNDEFN`, which creates a structured variable that contains the essential information about the function space from which approximants will be drawn. There are several pieces of information that must be specified and stored in the structure variable in order to define the function space: the type of basis function (e.g., Chebychev polynomial, spline, etc.), the number of basis functions, and the endpoints of the interpolation interval. If the approximant is multidimensional, the number of basis functions and the interval endpoints must be supplied for each dimension.

The function `FUNDEFN` defines the approximation function space using the syntax:

```
space = fundefn(bastype,n,a,b,order);
```

Here, on input, `bastype` is string referencing the basis function family, either 'cheb' for Chebychev polynomial basis or 'spli' for spline basis; `n` is the vector containing the degree of approximation along each dimension; `a` is the vector of left endpoints of interpolation intervals in each dimension; `b` is the vector of right endpoints of interpolation intervals in each dimension; and `order` is an optional input that specifies the order of the interpolating spline. On output, `space` is a structured Matlab variable containing numerous fields of information necessary for forming approximations in the chosen function space.

For example, suppose one wished to construct tenth degree Chebychev approximants for univariate functions defined on the interval $[-1, 2]$. Then

one would define the appropriate function space for approximation as follows:

```
space = fundefn('cheb',10,-1,2);
```

Suppose now that one wished to construct cubic spline approximants for bivariate functions defined on the two-dimensional interval $\{(x_1, x_2)|-1 \leq x_1 \leq 2, 4 \leq x_2 \leq 9\}$. Furthermore suppose that one wished to form an approximant using ten basis functions in the first direction and 15 basis functions in the second direction. Then one would issue the following command:

```
space = fundefn('spli',[10 15],[-1 2],[4 9]);
```

For spline interpolation, cubic (that is, third-order) spline interpolation is the default. However, other order splines may also be used for interpolation by specifying `order`. In particular, if one wished to construct linear spline approximants instead of cubic spline interpolants, one would issue the following command:

```
space = fundefn('spli',[10 15],[-1 2],[4 9],1);
```

Two procedures are provided for function approximation and simple data fitting. `FUNFITF` determines the basis coefficients of a member from the specified function space that approximates a given function f defined in an M-file or as an inline function. The syntax for this function approximation routine is:

```
c = funfitf(space,f,varargin);
```

Here, on input, `space` is the approximation function space defined using `FUNDEF`; `f` is the string name of the M-file or inline object that evaluates the function to be approximated; and `varargin` are additional parameters that are passed on to the function `f`. On output, `c` is the vector of basis function coefficients for the unique member of the approximating function space that interpolates the function f at the standard interpolation nodes associated with that space.

A second procedure, `FUNFITXY`, computes the basis coefficients of the function approximant that interpolates the values of a given function at arbitrary points that may, or may not, coincide with the standard interpolation nodes. The syntax for this function approximation routine is:

```
c = funfitxy(space,x,y);
```

Here, on input, `space` is the approximation function space defined using `FUNDEF`; `x` is the vector of points at which the function has been evaluated; and `y` is the vector of function values at those points. On output, `c` is the vector of basis function coefficients for the member of the approximating function space that interpolates f at the interpolation nodes supplied in x. If there are more data points than coefficients, `FUNFITXY` returns the least squares fit; the procedure can therefore be used for statistical data fitting as well as interpolation.

Once the approximant function space has been chosen and a specific approximant in that space has been selected by specifying the basis coefficients, then the procedure `FUNEVAL` may be used to evaluate the approximant at one or more points. The syntax for this function approximation routine is:

```
y = funeval(c,space,x);
```

Here, on input, `space` is the approximation function space defined using `FUNDEFN`; `c` is the vector of basis coefficients that identifies the approximant; and `x` is the point at which the approximant is to be evaluated, written as a 1 by d row vector. On output, `y` is the value of the approximant at x. If one wishes to evaluate the approximant at m points, then one may pass all these points to `FUNEVAL` at once as an m by d array `x`, in which case `y` is returned as an m by 1 vector of function values.

The procedure `FUNEVAL` may also be used to evaluate the derivatives or the approximant at one or more points. The syntax for evaluating derivatives is:

```
deriv = funeval(c,space,x,order);
```

were, on input, `order` is a 1 by d specifying the order of integration in each dimension. For example, to compute the first and second derivative of a univariate approximant, one issues the commands:

```
f1 = funeval(c,space,x,1);
f2 = funeval(c,space,x,2);
```

To compute the partial derivative of a bivariate approximant with respect to its first two arguments, one would issue the commands:

```
f1 = funeval(c,space,x,[1 0]);
f2 = funeval(c,space,x,[0 1]);
```

And to compute the second partial derivatives and the cross partial of a bivariate function, one would issue the commands:

```
f11 = funeval(c,space,x,[2 0]);
f22 = funeval(c,space,x,[0 2]);
f12 = funeval(c,space,x,[1 1]);
```

Some simple examples will help clarify how all of these procedures may be used to construct and evaluate function approximants. Suppose we are interested (for whatever reason) in approximating the univariate function

$$f(x) = \exp(-x)$$

on [-1,1]. The following script constructs the Chebychev approximant and then plots the errors using a finer grid than used in interpolation:

```
f     = inline('exp(-x)');
space = fundefn('cheb',10,-1,1);
c     = funfitf(space,f);
x     = nodeunif(1001,-1,1);
yact  = f(x);
yapp  = funeval(c,space,x);
plot(x,yact-yapp);
```

Here, we first define the function, `f`, using `inline`. Second, we use `FUNFITFN` to define the function space from which the approximant is to be drawn, in this case the space of 10 degree Chebychev polynomial approximants on [-1,1]. Third, we use `FUNFITF` to compute the coefficient vector for the approximant that interpolates the function at the standard Chebychev nodes. Fourth, we generate a fine grid of 1001 equally spaced nodes on the interval of interpolation and plot the difference between the actual function values `yact` and the approximated values `yapp`. The approximation error is plotted in Figure 6.9.

To other routines are useful in applied computational economic analysis. For many problems it is necessary to work directly with the basis matrices. For this purpose `FUNBAS` can be used. The command

```
B = funbas(space,x);
```

returns the matrix containing the values of the basis functions evaluated at the points x. The matrix containing the value of the basis functions associated with a derivative of given order at x may be retrieved by issuing the command

```
B = funbas(space,x,order);
```

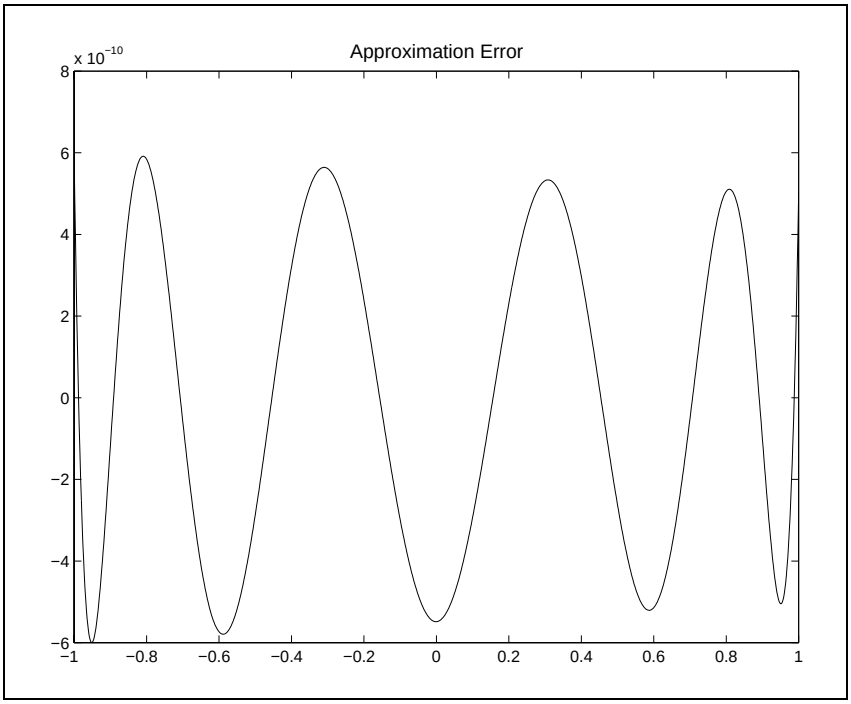

Figure 6.9

When a function is to be repeatedly evaluated at the same points but with different values of the coefficients substantial time saving are achieved by avoiding repeated recalculation of the basis. `FUNEVAL` therefore accepts a basis matrix as its second argument. The command

```
B = funbas(space,x);
y = funeval(c,B);
```

has the same effect as

```
y = funeval(space,x);
```

Finally, the procedure `FUNNODE` computes standard nodes for interpolation and function fitting. It returns a $1 \times d$ cell array associated with a specified coefficient structure. Its syntax is

```
x = FUNNODE(space);
```

6.7 Solving Functional Equations

In this section we consider a related but somewhat more difficult problem of solving functional equations. A general representation of the functional equation problem is to find a function f that satisfies

$$g(f, x) = 0 \text{ for } x \in [a, b].$$

Solving functional equations numerically involves finding a function $\hat{f}$ from a finite-dimensional function space that approximately satisfies $g(\hat{f}, x) = 0$. Again, it is useful to work with approximants that can be written in the form

$$f(x) \approx \hat{f}(x) = \sum c_j \phi_j(x),$$

where the ϕ_j are a set of basis functions. The condition to be satisfied can be written as

$$g(\sum c_j \phi_j, x) \approx 0 \text{ for } x \in [a, b].$$

The term $g(\sum c_j \phi_j, x)$can be thought of as a residual, which should be small (in some sense) by the choice of $\{c_j\}$. Notice that, for any choice of c, the residual is a function of x.

A general approach to solving functional equations numerically is *collocation*. The collocation strategy is to choose c in such a way as to make the residual zero at n prescribed nodes:

$$g(\sum c_j \phi_j, x_i) = 0 \text{ for } i = 1, 2, \ldots, n.$$

We now examine some examples of functional equations in Economics and demonstrate the use of collocation methods to solve them.

6.7.1 Cournot Oligopoly

In the standard microeconomic model of firm behavior, a firm facing a given cost function maximizes profit by setting marginal revenue (MR) equal to marginal cost (MC). The marginal cost is determined by the firm's technology and is a function of the amount of the good the firm produces (q). For a price taking firm, MR is simply the price the firm faces (p). An oligopolistic firm, however, recognizing that its actions affect price, takes the marginal

revenue to be $p + q\frac{dp}{dq}$. Of course the term $\frac{dp}{dq}$ is the problem. The Cournot assumption is that the firm acts as if any output change it makes will be unmatched by its competitors. This implies that

$$\frac{dp}{dq} = \frac{1}{D'(p)}$$

where $D(p)$ is the market demand for the good.

If we want to determine the effective supply for this firm at any given price, we need to find a function $q = S(p)$ that equates marginal cost with marginal revenue and therefore solves the functional equation:

$$p + \frac{S(p)}{D'(p)} - MC(S(p)) = 0$$

for all positive prices. In simple cases, this function can be found explicitly. For example, suppose that $MC(q) = c$ and $q = D(p) = p^{-\eta}$. It is easy to demonstrate that [3]

$$q = S(p) = \eta(p - c)p^{-\eta-1}.$$

With m identical firms, we can compute the (Cournot) equilibrium price for the whole industry by setting

$$mS(p) = D(p),$$

which, in the constant marginal cost case, yields

$$p = \left(\frac{1}{1 - \frac{1}{\eta m}}\right) c$$

(notice that this result produces the perfect competition result that $p = c$ as $m \to \infty$).

What are we to do, however, if the marginal cost function is not so nicely behaved? Suppose, for example, that

$$MC(q) = \alpha\sqrt{q} + q^2.$$

[3]Strictly speaking we should impose the $q \geq 0$ and write the residual as a complementarity (Kuhn-Tucker) condition. In $MC = c$ case this puts a kink at $p = c$, with $S(p) = 0$ for $p < c$.

Using the same demand function, the MR=MC condition becomes

$$\left(p - \frac{qp^{\eta+1}}{\eta}\right) - (\alpha\sqrt{q} + q^2) = 0.$$

There is no way to find an explicit expression for $q = S(p)$ from this relationship.

To find a solution we must resort to numerical methods, finding a function $\hat{S}$ that approximates S over some interval $p \in [a, b]$. Using collocation, we define a set of price nodes (p) and an associated basis matrix B. These are used in a function that, given a coefficient vector c, computes the residual equation at the price nodes. This function is then passed to a root finding algorithm. The following script demonstrates how to perform these tasks:

```
alpha=1; eta=1.5;
n=25; a=0.1; b=3;
space = FUNDEFN('cheb',n,a,b);
p = FUNNODE(space);
B = FUNBAS(space,p);
c = B\sqrt(p);
c = broyden('resid',c,[],p,alpha,eta,B);
```

The script calls a function 'resid' the computes the functional equation residual for any choice of coefficient vector c:

```
function resid=f(c,p,alpha,eta,B);
dp    = (-1./eta)*p.^(eta+1);
q     = B*c;
resid = p + q.*dp - alpha*sqrt(q) - q.^2;
```

The resulting coefficients, `c`, can then be used to evaluate the "supply" functions. A set of industry "supply" functions and the industry demand function for $\alpha = 1$, $\eta = 1.5$ are illustrated in Figure 6.10. The equilibrium price is determined by the intersection of the industry "supply" and demand curves. A plot of the equilibrium price for alternative industry sizes is shown in Figure 6.11.

It should be emphasized that all collocation problems involve writing a function to compute the residuals. This function is passed to a root-finding algorithm (unless the problem is linear). Typically, however, it makes sense to initialize certain variables, such as the basis matrices needed to evaluate the residual function, as well as any other variables whose value does not depend on the coefficient values. Thus there are typically two procedures needed to solve collocation problems. The first sets up the problem and initializes

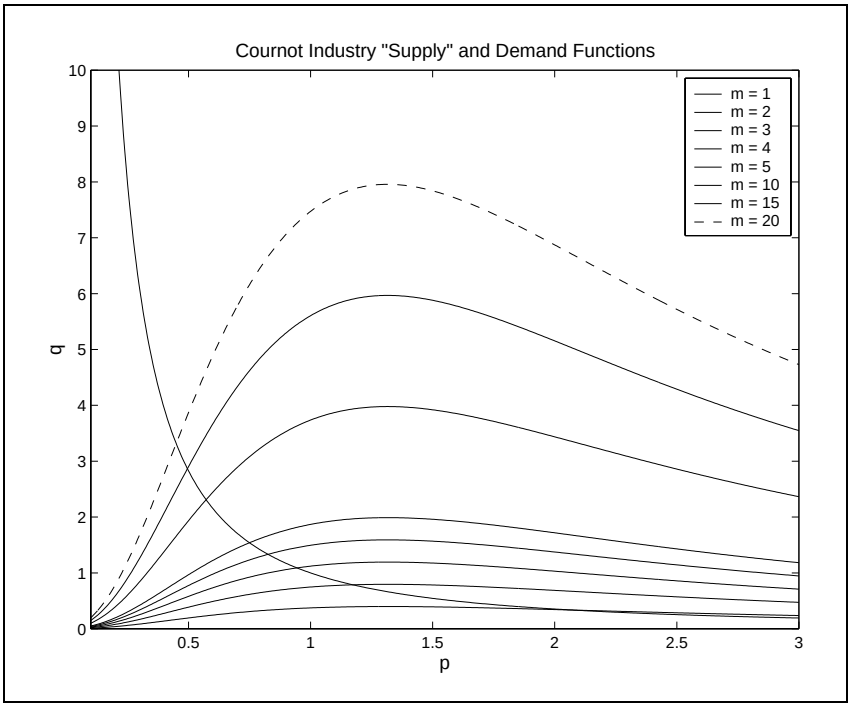

Figure 6.10

variables. It then call a root-finding algorithm, passing it the name of the second procedure, which computes the residuals. In our example we have combined these into a single Matlab function to reduce the number of files needed to solve the problem.

It is also generally a good idea to implement an additional step in solving any collocation problem to analyze how well the problem has been solved. Although we generally do not know the true solution, we can compute the value of the residual at any particular point. If the input argument is low-dimensional (1 or 2) we can plot the residual function at a grid of points, with the grid much finer than that used to define the collocation nodes. Even if plotting is infeasible, one can still evaluate the residual function at a grid of points and determine the maximum absolute residual or the mean squared residual. This should give you a reasonable idea of how well the approximation solves the problem. Residuals for the Cournot example can be plotted against price with the following script:

```
p = nodeunif(501,a,b)';
```

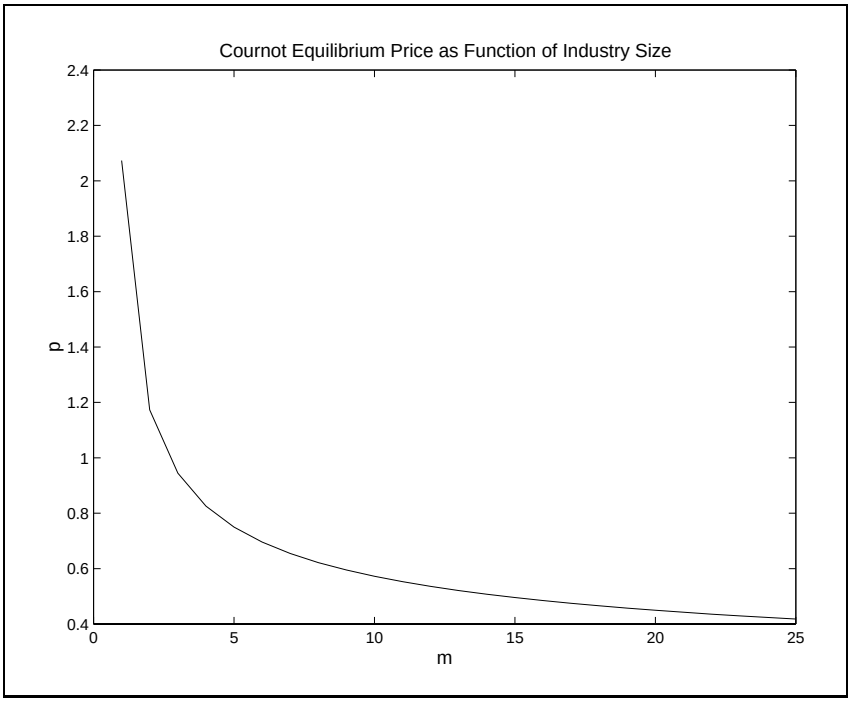

Figure 6.11

```
B = FUNBAS(space,p);
resid = f(c,p,alpha,eta,B);
plot(p,resid)
```

The result is shown in Figure 6.12, which makes clear that the approximation adequately solves the functional equation.

6.7.2 Function Inverses

As another example, consider the problem of inverting a function g. Specifically, we would like to approximate a function $f(x)$ that satisfies $g(f(x)) = x$ on some interval $a \leq x \leq b$. The residual function here is simply $r(x) = g(f(x)) - x$. The collocation approach is therefore to find the c that satisfies

$$g\left(\sum_j c_j \phi_j(x_i)\right) - x_i = 0$$

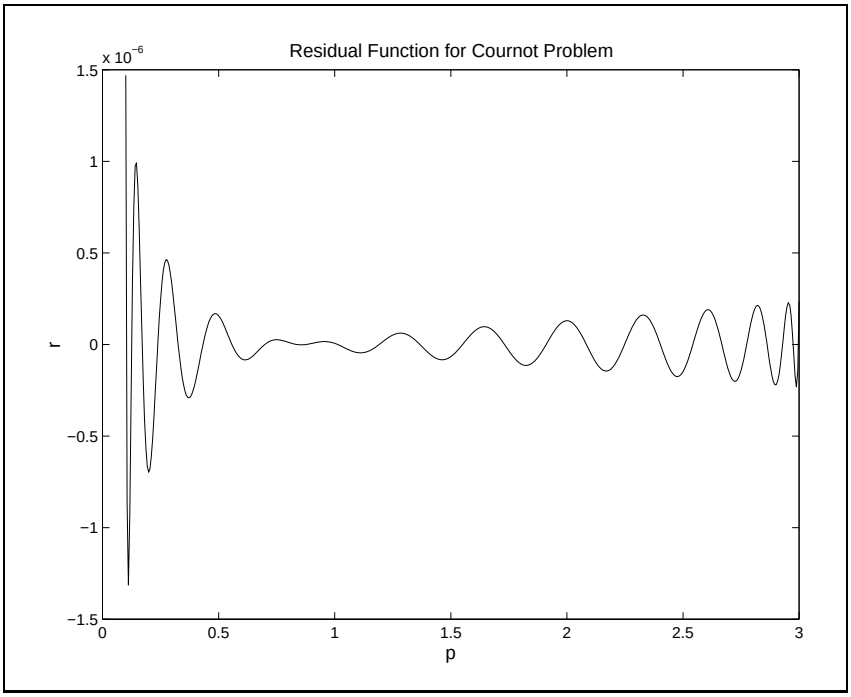

Figure 6.12

at a selected set of x_i. Except in the trivial case in which g is linear, c must be found using a non-linear root finding algorithm.

It is straightforward to write a utility procedure to find the inverse of an arbitrary function. To accomplish this we will want to define a set of x values for collocation nodes and form a basis matrix at those values. These will be predefined and stored in memory in the initialization phase. If initial coefficient values are not passed to the function, we should use some reasonable default values. It is not clear how to what good values would be so we've simply defined an identity mapping, $f(x) = x$, as our initial guess. This works fine for our example below; if this doesn't work for the function of your choice, you'll have to come up with a better initial guess.

To illustrate, suppose you want to approximate the inverse of $\exp(y)$ over some range such as $x \in [1, 2]$. We must find a function f for which it is approximately true that $\exp(f(x)) - x = 0$ for $x \in [1, 2]$. The following script computes an approximate inverse via collocation:

```
space = FUNDEFN('cheb',6,1,2);
```

```
x = FUNNODE(space);
c = FUNFITF(space,inline('x'));
c = BROYDEN('f',c,[],space,x);
```

The script calls a function 'resid' the computes the functional equation residual for any choice of coefficient vector c:

```
function resid=resid(c,space,x)
resid = exp(funeval(c,space,x))-x;
```

The following script generates a plot of the residual function, shown in Figure 6.13, and a plot of the true approximation error, shown in Figure 6.14:

```
xplot = nodeunif(101,1,2);
figure(1)
plot(xplot,exp(FUNEVAL(c,space,xplot))-xplot)
title('Residual Function: exp(f(x))-x');
xlabel('x'); ylabel('r')
figure(2)
plot(xplot,log(xplot)-FUNEVAL(c,space,xplot))
title('Approximation Errors for exp^{-1}(x)');
xlabel('x');ylabel('error')
```

Even with only 6 nodes, it is clear that we have found a good approximation to the inverse. Of course we know that the inverse is $\ln(x)$, which allowed us to compute the directly how well we have done.

6.7.3 Linear First Order Differential Equations

Consider the first order, linear differential equation with a non-constant coefficients

$$f'(x) - \alpha_1(x) - \alpha_2(x)f(x) = 0.$$

An approximate solution can be expressed as the linear relationship

$$\sum_j \left(\phi_j'(x_i) - \alpha_1 - \alpha_2(x_i)c_j\phi_j(x_i)\right) = 0,$$

for some specified set of x_i.

To obtain a specific solution to this problem, however, one additional restriction must be imposed. This will generally be an initial condition of the form condition of the form

$$f(a) = k$$

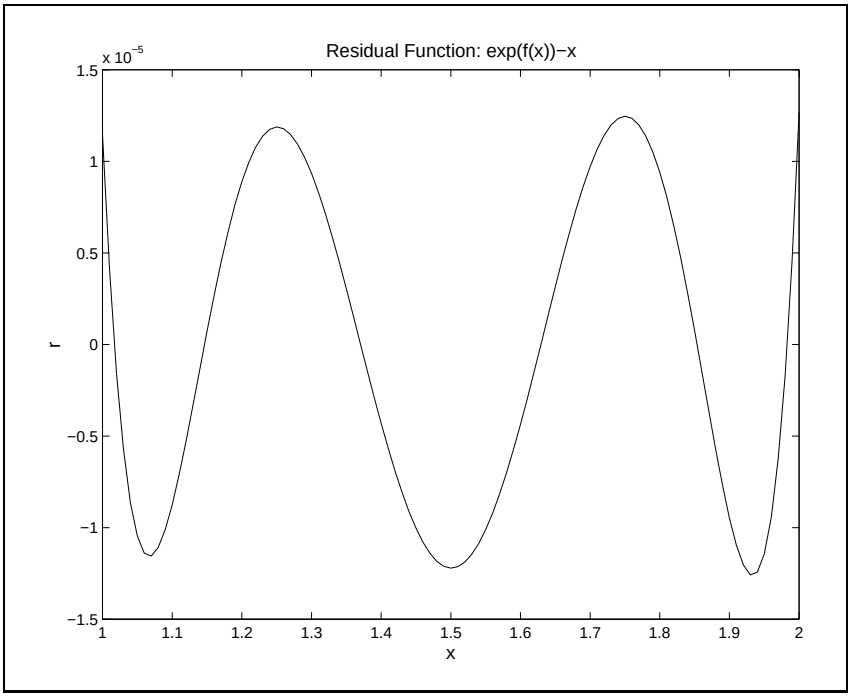

Figure 6.13

for some specified constant k. This restriction can be imposed on the approximating function:

$$\sum_j c_j \phi_j(a) = k.$$

Given an n dimensional basis and $n-1$ nodes, this results in a linear equation system that can be solved by matrix inversion.

A general solver for the linear differential equation is:

```
function [c,space]=LNDIFFEQ(n,a,b,alpha,k)
x  = chebnode(n-1,a,b);              % n-1 Chebyshev nodes
B  = chebbas(n,a,b,x);               % basis matrix for the DE
dB = chebbas(n,a,b,x,1);             % derivative of basis matrix
A  = feval(alpha,x);                 % get the alpha values
A1 = A(:,1)*ones(1,n);               % expand the alpha values
A2 = A(:,2)*ones(1,n);
B  = dB-A1-A2.*B;                    % residual function
B  = [B;chebbas(n,a,b,a)];           % append the boundary condition
```

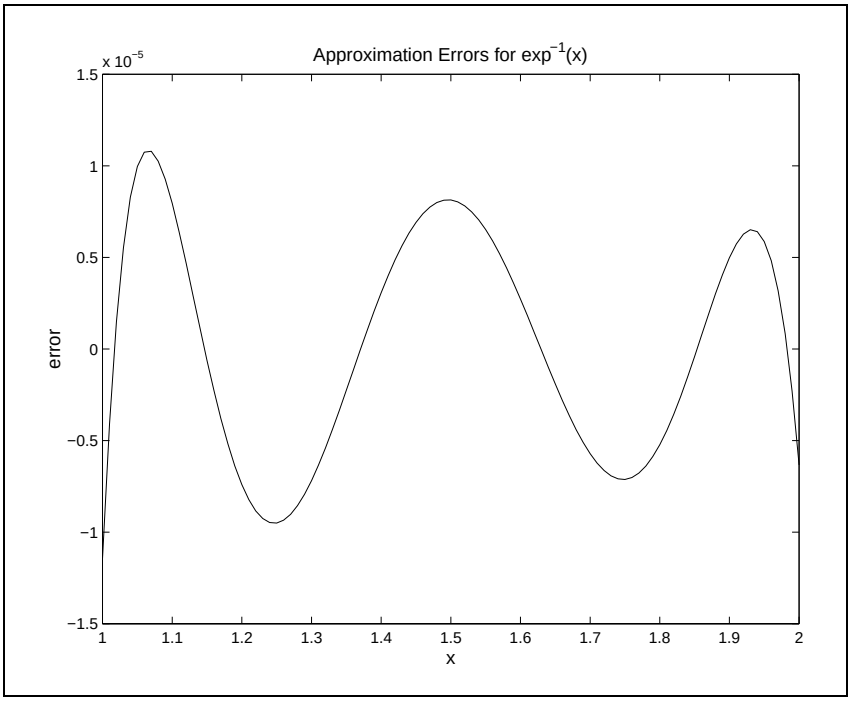

Figure 6.14

```
c  = B\[zeros(n-1,1);k];              % computes the coefficient values
space = fundefn('cheb',n,a,b);        % creates function structure
```

To illustrate its use, consider the differential equation

$$f'(x) = f(x)$$

on $[0, 1]$, with $f(0) = 1$ (the exact solution is $\exp(x)$). The script below solves the equation numerically using a 10 node approximation and then plots out the residual function and the approximation errors. The resulting plots are shown in Figures 6.15 and 6.16.

```
alpha=inline('ones(size(x,1),1)*[0 1]','x');
[c,space]=LNDIFFEQ(10,0,1,alpha,1);
x=nodeunif(301,0,1);
figure(1)
plot(x,FUNEVAL(c,space,x,1)-FUNEVAL(c,space,x));
title('Residual Function for f''(x)-f(x)')
xlabel('x');ylabel('r')
```

```
figure(2)
plot(x,exp(x)-FUNEVAL(c,space,x))
title('Approximation Error: exp(x)-f(x)')
xlabel('x');ylabel('error')
```

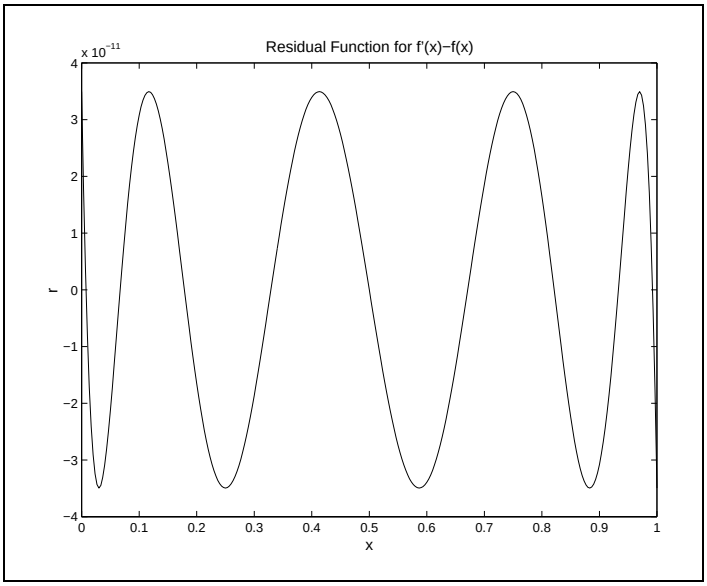

Figure 6.15

Exercises

1. In the Cournot model each firm takes the output of the other firms as given when determining its output level. An alternative assumption is that each firm takes the output decision function as given when making its own output choice. This can be expressed as the assumption that

$$\frac{dp}{dq_i} = \frac{1}{D'(p)} \sum_{j=1}^{n} \frac{dq_j}{dq_i} = \frac{1}{D'(p)} \left(1 + \sum_{j \neq i} \frac{dS_j(p)}{dp} \frac{dp}{dq_i} \right).$$

   Solving this for dp/dq_i yields

$$\frac{dp}{dq_i} = \frac{1}{D'(p) - \sum_{j \neq i} S_j'(p)}.$$

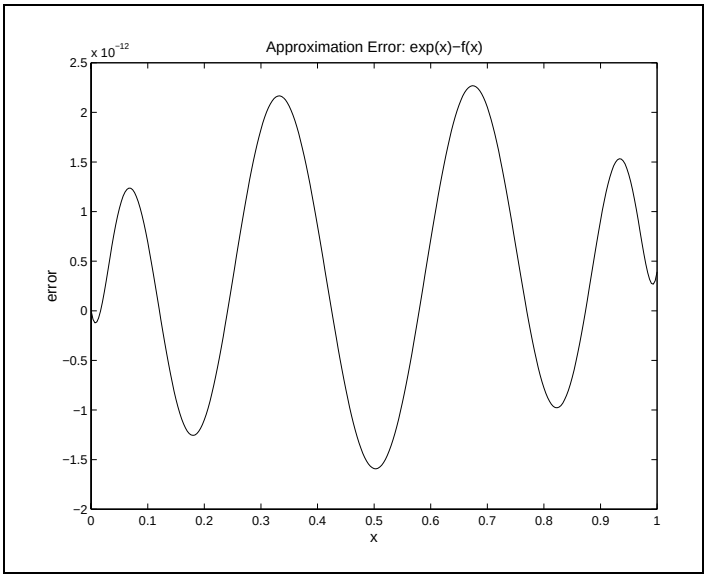

Figure 6.16

In an identical firm industry this means that each firm assumes the other firms will react in the same way it would so this expression simplifies to

$$\frac{dp}{dq} = \frac{1}{D'(p) - (n-1)S'(p)}.$$

This expression differs from the Cournot case in the extra term in the denominator (which is 0 in the monopoly situation of $n = 1$).

Write a function to solve this problem analogous to the one in this Chapter and a demo file to produce the analogous plots. The function must take the parameters n (industry size), in contrast to the Cournot case, and must compute the derivative of the $q = S(p)$ function to compute the residual function.

2. The least absolute deviation fit of a function that is linear in its coefficients solves

$$\min_c \sum_{i=1}^{n} |y_i - \phi(x_i)c|.$$

Defining Φ to be the basis matrix formed using all values of the x_i and y the vector of the y_i values, this can be written as the following linear program

$$\min_{c,e^+,e^-} (e^+ + e^-)^\top \underline{1}$$

s.t.

$$\Phi c + e^+ - e^- = y.$$

Write a Matlab function MAD (for minimum absolute deviation, of course) analogous to the MINMAX procedure defined in the appendix.

3. Construct the 5- and 50-degree approximants for the function $f(x) = \exp(-x^2)$ on the interval $[-1, 1]$ using each of the interpolation schemes below. For each scheme and degree of approximation, estimate the sup norm approximation error by computing the maximum absolute deviation between the function and approximant at 100 evenly spaced points. Also, graph the approximation error for the degree 5 approximant.

   (a) Uniform node, monomial basis polynomial approximant

   (b) Chebychev node, Chebychev basis polynomial approximant

   (c) Uniform node, linear spline approximant

   (d) Uniform node, cubic spline approximant

4. Consider the potato market model discussed in the Chapter 3 problems. Construct a 5^{th} degree Chebychev polynomial approximant for the function relating the period 1 price to initial supply s over the interval $s \in [1, 3]$. Interpolate the polynomial at $s = 1$, $s = 2$, and $s = 3$ and compare to the interpolated values to those obtained earlier.

5. Consider again the potato market model. Assume now that supply s is the product of acreage a and yield y where yield can achieve one of two equiprobable outcomes, a low yield 0.75 and a high yield 1.25, and that acreage is a function of the price expected in the harvest period:

$$a = 0.5 + 0.5Ep_1.$$

The rational expectations equilibrium acreage level and expected price satisfy the acreage supply function and

$$Ep_1 = 0.5f(0.75a) + 0.5f(1.25a)$$

where f is the function approximated in the preceding problem. Compute the rational expectations equilibrium of the model using the 10^{th} degree Chebychev polynomial approximation for f computed in the preceding problem.

6. With the basis functions of your choice use collocation to numerically solve the following differential equation for $x \in [0, 1]$:

$$(1 + x^2)v(x) - v''(x) = x^2,$$

with $v(0) = v(1) = 0$. Plot the residual function to ensure that the maximum value of the residual is less than 1e-8.

Chapter 7

Discrete Time Discrete State Dynamic Models

With this chapter, we begin our study of dynamic economic models. Dynamic economic models often present three complications rarely encountered together in dynamic physical science models. First, humans are cogent, future-regarding beings capable of assessing how their actions will affect them in the future as well as in the present. Thus, most useful dynamic economic models are future-looking. Second, many aspects of human behavior are unpredictable. Thus, most useful dynamic economic models are inherently stochastic. Third, the predictable component of human behavior is often complex. Thus, most useful dynamic economic models are inherently nonlinear.

The complications inherent in forward-looking, stochastic, nonlinear models make it impossible to obtain explicit analytic solutions to dynamic economic models. However, the proliferation of affordable personal computers, the phenomenal increase of computational speed, and developments of theoretical insights into the efficient use of computers over the last two decades now make it possible for economists to analyze dynamic models much more thoroughly using numerical methods.

The next three chapters are devoted to the numerical analysis of dynamic economic models in discrete time and are followed by three chapters on dynamic economic models in continuous time. In this chapter we study the most simple of these models: the discrete time, discrete state Markov decision model. Though the model is simple, the methods used to analyze the model lay the foundations for the methods developed in subsequent chapters

to analyze more complicated models with continuous states and time.

7.1 Discrete Dynamic Programming

The discrete time, discrete state Markov decision model has the following structure: in every period t, an agent observes the state of an economic process s_t, takes an action x_t, and earns a reward $f(x_t, s_t)$ that depends on both the state of the process and the action taken. The state space S, which enumerates all the states attainable by the process, and the action space X, which enumerates all actions that may be taken by the agent, are both finite. The state of the economic process follows a controlled Markov probability law. That is, the distribution of next period's state, conditional on all currently available information, depends only on the current state of the process and the agent's action:

$$\Pr(s_{t+1} = s' | x_t = x, s_t = s, \text{ other information at } t\ ) = P(s'|x, s).$$

The agent seeks a policy $\{x_t^*\}_{t=1}^T$ that prescribes the action $x_t = x_t^*(s_t)$ that should be taken in each state so as to maximize the present value of current and expected future rewards over time, discounted at a per-period factor $\delta \in (0, 1]$:

$$\max_{\{x_t^*\}_{t=0}^T} E\left[\sum_{t=0}^{T} \delta^t f(x_t, s_t)\right].$$

A discrete Markov decision model may have an infinite horizon ($T = \infty$) or a finite horizon ($T < \infty$). The model may also be either deterministic or stochastic. It is deterministic if next period's state is known with certainty once the current period's state and action are known. In this case, it is beneficial to dispense with the probability transition law as a description of how the state evolves and use instead a deterministic state transition function g, which explicitly gives the state transitions:

$$s_{t+1} = g(x_t, s_t).$$

Discrete Markov decision models may be analyzed and understood using the dynamic programming principles developed by Richard Bellman (1956). Dynamic programming is an analytic approach in which a multiperiod model

is effectively decomposed into a sequence two period models. Dynamic programming is based on the Principle of Optimality, which was articulated by Bellman as follows:

> "An optimal policy has the property that, whatever the initial state and decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision."

The Principle of Optimality can be formally expressed in terms of the value functions V_t. For each period t and state s, $V_t(s)$ specifies the maximum attainable sum of current and expected future rewards, given that the process is in state s and the current period is t. Bellman's Principle implies that the value functions must satisfy Bellman's recursion equation

$$V_t(s) = \max_{x \in X(s)} \{f(x,s) + \delta \sum_{s' \in S} P(s'|x,s) V_{t+1}(s')\} \quad s \in S.$$

Bellman's equation captures the essential problem faced by a dynamic, future-regarding optimizing agent: the need to balance the immediate reward $f(x_t, s_t)$ with discounted expected value of future rewards $\delta E_t V_{t+1}(s_{t+1})$. Given the value functions, the optimal policies $x_t^*(s)$ are simply the solutions to the optimization problems embedded in Bellman's equation.

In a finite horizon model, we adopt the convention that the optimizing agent faces decisions up to and including a final decision period $T < \infty$. The agent faces no decisions after the terminal period T, but may earn a final reward V_{T+1} in period $T+1$. The terminal value is typically fixed by some economically relevant terminal condition. In many applications, V_{T+1} is identically zero, indicating that no rewards are earned by the agent beyond the terminal decision period. In other applications, V_{T+1} may specify a salvage value earned by the agent after making his final decision in period T.

For the finite horizon discrete Markov decision model to be well posed, the terminal value V_{T+1} must be specified by the analyst. Given the terminal value function, the finite horizon decision model in principle may be solved recursively by repeated application of Bellman's equation: having $V_{T+1}(s)$, solve for $V_T(s)$ for all states s; having V_T, solve for $V_{T-1}(s)$ for all states s; having V_{T-1}, solve for $V_{T-2}(s)$ for all states s; and so on. The process continues until $V_0(s)$ is derived for all states s. Because only finitely many actions

are possible, the optimization problem embedded in Bellman's equation can always be solved by performing finitely many arithmetic operations. Thus, the value functions of a finite horizon discrete Markov decision model are always well-defined, although in some cases more than one policy of state-contingent actions may yield the maximum expected stream of rewards (i.e., the optimal action may not be unique).

If the decision problem has an infinite horizon, the value will not depend on time t and will be the same in every period; we may, therefore, drop the time subscripts and simply denote the common value function by V. Bellman's equation therefore becomes the fixed-point equation

$$V(s) = \max_{x \in X(s)} \left[f(x,s) + \delta \sum_{s' \in S} P(s'|x,s) V(s') \right], \quad s \in S.$$

If the discount factor δ is less than one, the mapping underlying Bellman's equation is a strong contraction. The Contraction Mapping Theorem thus guarantees the existence and uniqueness of the infinite horizon value function.[1]

7.2 Economic Examples

Specification of a discrete Markov decision model requires several pieces of information: the reward and state transition or transition probabilities associated with each state and action, the discount factor δ, the time horizon T, and, if the model has finite horizon, the terminal value V_{T+1}. This section provides seven economic examples that illustrate how the necessary information is specified and how the Bellman equation is formulated.

7.2.1 Mine Management

A mine operator must determine the optimal ore extraction schedule for a mine that will be shut down and abandoned after T years. The market price of one ton of ore is p and the total cost of extracting x tons of ore in any

[1] Value functions in infinite horizon problems could be time dependent if f, P, or δ displayed time dependence. However, this creates difficulties in developing solution methods, and we have chosen not to explicitly consider this possibility. Fortunately, most infinite horizon economic model do not display such time dependence.

year is $c = x^2/(1+s)$ where s is the stock of ore available at the beginning of the year in tons. The mine currently contains $\bar{s}$ tons of ore. If the tons of ore extracted in any year must be an integer, what production schedule maximizes profits?

This is a finite horizon, deterministic model with time $t = \{1, 2, \ldots, T\}$ measured in years. The state is

$$\begin{aligned} s &= \text{stock of ore in tons} \\ s &\in S = \{0, 1, 2, \ldots, \bar{s}\}; \end{aligned}$$

the action is

$$\begin{aligned} x &= \text{ore extracted in tons} \\ x &\in X(s) = \{0, 1, 2, \ldots, s\}; \end{aligned}$$

the state transition function is

$$s' = g(s, x) = s - x;$$

and the reward function is

$$f(s, x) = px - x^2/(1+s).$$

The value function

$$V_t(s) = \text{value of mine with } s \text{ tons of ore at } t$$

satisfies Bellman's equation:

$$V_t(s) = \max_{x \in \{0,1,2,\ldots,s\}} \{px - x^2/(1+s) + \delta V_{t+1}(s-x)\}, \qquad s \in S.$$

subject to the terminal condition

$$V_{T+1}(s) = 0, \qquad s \in S.$$

7.2.2 Deterministic Asset Replacement

At the end of each lactation cycle a dairy producer must decide whether to keep a cow again or replace it with a new one. A cow yields $y(s)$ tons of milk over lactation cycle s, up to ten lactations. Upon completion of the 10^{th} lactation, a cow becomes unproductive and must be replaced. The net cost of replacing a cow is c dollars and the profit contribution of milk is p dollars per ton. What replacement policy maximizes profits?

This is an infinite horizon, deterministic model with time t measured in lactation cycles. The state is

$$\begin{aligned} s &= \text{lactation number of cow} \\ s &\in S = \{1, 2, \ldots, 10\}; \end{aligned}$$

the action is

$$\begin{aligned} x &= \text{replacement decision} \\ x &\in X(s) = \begin{cases} \{\text{keep}, \text{replace}\} & s < 10 \\ \{\text{replace}\} & s = 10 \end{cases}; \end{aligned}$$

the state transition function is

$$s' = g(x, s) = \begin{cases} s + 1 & x = \text{keep} \\ 1 & x = \text{replace} \end{cases};$$

and the reward function is

$$f(x, s) = \begin{cases} py(s) & x = \text{keep} \\ py(s) - c & x = \text{replace}. \end{cases}$$

The value function

$$V(s) = \text{value of cow entering lactation cycle } s$$

must satisfy Bellman's equation

$$V(s) = \begin{cases} \max\{py(s) + \delta V(s+1), py(s) - c + \delta V(1)\}, & s < 10, \\ py(s) - c + \delta V(1), & s = 10. \end{cases}$$

Bellman's equation asserts that if we keep a cow after lactation cycle s, we receive net earnings $py(s)$ during that lactation and begin the subsequent cycle with a cow worth $V(s+1)$; if we replace the cow after lactation s, on the other hand, we receive net earnings of $py(s) - c$ during that lactation

cycle and begin the subsequent cycle with a cow worth $V(1)$. Actually, our language is a little loose here. The value function measures not only the value of the current cow in cycle s, but also the additional value of all future cows that will replace her. It would therefore be more correct to say that $V(s)$ measures the value of having a cow in cycle s.

7.2.3 Stochastic Asset Replacement

Suppose now that dairy cows vary in productivity. Specifically, each cow belongs to one of n productivity classes, denoted $u \in \{1, 2, \ldots, n\}$. A cow in productivity class u yields $q_u y(s)$ tons of milk over lactation cycle s, where q_u is a quality multiplier and $y(s)$ is the industry average yield. When replacing a dairy cow, the farmer will not know how productive the new cow will be until the end of its first lactation. Cows of quality class u are obtained from the replacement pool with probability w_u. What is the optimal lactation-replacement policy?

This is an infinite horizon, stochastic model with time t measured in lactation cycles. The two-dimensional state is

$$\begin{aligned} s &= \text{lactation number of cow} \\ s &\in S_1 = \{1, 2, \ldots, 10\} \end{aligned}$$

and

$$\begin{aligned} u &= \text{quality class of cow} \\ u &\in S_2 = \{1, 2, \ldots, n\}; \end{aligned}$$

the action is

$$\begin{aligned} x &= \text{replacement decision} \\ x &\in X(s) = \begin{cases} \{\text{keep}, \text{replace}\} & s < 10 \\ \{\text{replace}\} & s = 10 \end{cases} \end{aligned}$$

The state transition probability rule is

$$P(s', u'|x, s, u) = \begin{cases} 1 & s' = s + 1, u' = u, x = \text{keep} \\ w_{u'} & s' = 1, x = \text{replace} \\ 0 & \text{otherwise.} \end{cases}$$

and the reward function is

$$f(x, s, u) = \begin{cases} pq_u y(s) & x = \text{keep} \\ pq_u y(s) - c & x = \text{replace.} \end{cases}$$

The value function

$V(s,u)$ = value of having a cow of quality q_u entering lactation s

must satisfy Bellman's equation

$$V(s,u) = \max\{pq_u y(s) + \delta V(s+1,u), pq_u y(s) - c + \delta \sum_{u' \in S_2} w_{u'} V(1,u')\},$$

for $s < 10$ and

$$V(s,u) = pq_u y(s) - c + \delta \sum_{u' \in S_2} w_{u'} V(1,u')$$

for $s = 10$.

7.2.4 Option Pricing

An American put option gives the holder the right, but not the obligation, to sell a specified quantity of a commodity at a specified strike price on or before a specified expiration date. In the Cox-Ross-Rubinstein binomial option pricing model, the price of the commodity is assumed to follow a two-state discrete jump process. Specifically, if the price of the commodity is p in period t, then its price in period $t+1$ will be pu with probability q and p/u with probability $1-q$ where:

$$\begin{aligned} u &= \exp(\sigma\sqrt{\Delta t}) > 1 \\ q &= \tfrac{1}{2} + \frac{\sqrt{\Delta t}}{2\sigma}\left(r - \tfrac{1}{2}\sigma^2\right) \\ \delta &= \exp(-r\Delta t). \end{aligned}$$

Here, r is the annualized interest rate, continuously compounded, σ is the annualized volatility of the commodity price, and Δt is the length of a period in years. Assuming the current price of the commodity is p_0, what is the value of an American put option if it has a strike price $\bar{p}$ and if it expires T years from today?

This is a finite horizon, stochastic model where time $t \in \{0,1,2,\ldots,N\}$ is measured in periods of length $\Delta t = T/N$ years each. The state is[2]

$$\begin{aligned} p &= \text{commodity price} \\ p &\in S = \{p_1 u^i | i = -N-1, -N, \ldots, N, N+1\}. \end{aligned}$$

[2]In this example, we alter our notation to conform with standard treatments of option valuation. Thus, the state is the price, denoted by p, the number of time periods until expiration is N, and T reserved for the time to expiration (in years).

The action is

$$\begin{aligned} x &= \text{decision to keep or exercise} \\ x &\in X = \{\text{keep}, \text{exercise}\}; \end{aligned}$$

the state transition probability rule is

$$P(p'|x,p) = \begin{cases} q & p' = pu \\ 1-q & p' = p/u \\ 0 & \text{otherwise} \end{cases}$$

the reward function is

$$f(p,x) = \begin{cases} 0 & x = \text{keep} \\ \bar{p} - p & x = \text{exercise} \end{cases}$$

The value function

$$V_t(p) = \text{option value at } t, \text{ if commodity price is } p,$$

must satisfy Bellman's equation

$$V_t(p) = \max\{\ \bar{p} - p,\ q\delta V_{t+1}(pu) + (1-q)\delta V_{t+1}(p/u)\ \}$$

subject to the post-terminal condition

$$V_{N+1}(p) = 0$$

Note that if the option is exercised, the owner receives $\bar{p} - p$. If he does not exercise the option, however, he earns no immediate reward but will have an option in hand the following period worth $V_{t+1}(pu)$ with probability q and $V_{t+1}(p/u)$ with probability $1-q$. In option expires in the terminal period, making it valueless the following period; as such, the post-terminal salvage value is zero.

7.2.5 Job Search

At the beginning of each week, an infinitely-lived worker finds himself either employed or unemployed and must decide whether to be active in the labor market over the coming week by working, if he is employed, or by searching for a job, if he is unemployed. An active employed worker earns a wage w. An active unemployed worker earns an unemployment benefit u. An inactive

worker earns a psychic benefit v from additional leisure, but no income. An unemployed worker that looks for a job will find one with probability p by the end of the week. An employed worker that remains at his job will be fired with probability q at the end of the week. What is the worker's optimal labor policy?

This is a infinite horizon, stochastic model with time $t = \{1, 2, \ldots, \infty\}$ measured in weeks. The state is

$$\begin{aligned} s &= \text{employment state} \\ s &\in S = \{\text{unemployed}(0), \text{employed}(1)\} \end{aligned}$$

and the action is

$$\begin{aligned} x &= \text{labor force participation decision} \\ x &\in X = \{\text{inactive}(0), \text{active}(1)\}. \end{aligned}$$

The state transition probability rule is

$$P(s'|s,x) = \begin{cases} 1 & x = 0, s' = 0 & \text{(inactive worker)} \\ 1-p & x = 1, s = 0, s' = 0 & \text{(searches, finds no job)} \\ p & x = 1, s = 0, s' = 1 & \text{(searches, finds job)} \\ q & x = 1, s = 1, s' = 0 & \text{(works, loses job)} \\ 1-q & x = 1, s = 1, s' = 1 & \text{(works, keeps job)} \\ 0 & \text{otherwise;} & \end{cases}$$

and the reward function is

$$f(s,x) = \begin{cases} v & x = 0 & \text{(inactive, receives leisure)} \\ u & x = 1, s = 0 & \text{(searching, receives benefit)} \\ w & x = 1, s = 1 & \text{(working, receives wage)} \end{cases}$$

The value function

$$V(s) = \text{Value of being in employment state } s \text{ at beginning of week,}$$

must satisfy Bellman's equation

$$V(s) = \begin{cases} \max\{v + \delta V(0), u + \delta p V(1) + \delta(1-p)V(0)\}, & s = 0 \\ \max\{v + \delta V(0), w + \delta q V(0) + \delta(1-q)V(1)\}, & s = 1 \end{cases}$$

7.2.6 Optimal Irrigation

Water from a dam can be used for either irrigation or recreation. Irrigation during the spring benefits farmers, but reduces the dam's water level during the summer, damaging recreational users. Specifically, farmer and recreational user benefits in year t are, respectively, $F(x_t)$ and $G(y_t)$, where x_t are the units of water used for irrigation and y_t are the units of water remaining for recreation. Water levels are replenished by random rainfall during the winter. With probability p, it rains one unit; with probability $1-p$ is does not rain at all. The dam has a capacity of M units of water and excess rainfall flows out of the dam without benefit to either farmer or recreational user. Derive the irrigation flow policy that maximizes the sum of farmer and recreational user benefits over an infinite time horizon.

This is a infinite horizon, stochastic model with time $t = \{1, 2, \ldots, \infty\}$ measured in years. The state is

$$\begin{aligned} s &= \text{units of water in dam at beginning of year} \\ s &\in S = \{0, 1, 2, \ldots, M\} \end{aligned}$$

and

$$\begin{aligned} x &= \text{units of water released for irrigation during year} \\ x &\in X(s) = \{0, 1, 2, \ldots, s\}. \end{aligned}$$

The state transition probability rule is

$$P(s'|s,x) = \begin{cases} p & s' = \min(s - x + 1, M) \quad \text{(rain)} \\ 1-p & s' = s - x, \quad \text{(no rain)} \\ 0 & \text{otherwise} \end{cases}$$

and the reward function is

$$f(s,x) = F(x) + G(s-x).$$

The value function

$$V(s) = \text{Value of } s \text{ units of water in dam at beginning of year } t.$$

must satisfy Bellman's equation:

$$V(s) = \max_{x=0,1,\ldots,s} \{f(s,x) + \delta p V(\min(s - x + 1, M)) + \delta(1-p)V(s-x)\}.$$

7.2.7 Bioeconomic Model

In order to survive, an animal must forage for food in one of m distinct areas. In area x, the animal survives predation with probability p_x, finds food with probability q_x, and, if it finds food, gains e_x energy units. The animal expends one energy unit every period and has a maximum energy carrying capacity $\bar{s}$. If the animal's energy stock drops to zero, it dies. What foraging pattern maximizes the animal's probability of surviving T years to reproduce at the beginning of period $T+1$?

This is a finite horizon, stochastic model with time $t = \{1, 2, \ldots, T\}$ measured in foraging periods. The state is

$$\begin{aligned} s &= \text{stock of energy} \\ s &\in S = \{0, 1, 2, \ldots, \bar{s}\}; \end{aligned}$$

the action is

$$\begin{aligned} x &= \text{foraging area} \\ x &\in X = \{1, 2, \ldots, m\}. \end{aligned}$$

The state transition probability rule is, for $s = 0$,

$$P(s'|s,x) = \begin{cases} 1 & s' = 0 \qquad \text{(death is permanent)} \\ 0 & \text{otherwise;} \end{cases}$$

and, for $s > 0$,

$$P(s'|s,x) = \begin{cases} p_x q_x & s' = \min(\bar{s}, s-1+e_x) & \text{(survive, finds food)} \\ p_x(1-q_x) & s' = s-1 & \text{(survive, no food)} \\ (1-p_x) & s' = 0 & \text{(does not survive)} \\ 0 & \text{otherwise.} & \end{cases}$$

The reward function is

$$f(s,x) = 0.$$

Here, $s = 0$ is an absorbing state that, once entered, is never exited. More to the point, an animal whose energy stocks fall to zero dies, and remains dead. The reward function for periods 1 through T is zero, because there is only one payoff, surviving to procreate, and this payoff is earned in period $T+1$.

The value function

$$V_t(s) = \text{probability of procreating, given energy stocks } s \text{ in period } t$$

must satisfy Bellman's equation

$$V_t(s) = \max_{x \in X} \{p_x q_x V_{t+1}(\min(\bar{s}, s - 1 + e)) + p_x(1 - q_x)V_{t+1}(s - 1)\},$$

for $t \in 1, \ldots, T$, with $V_t(0) = 0$, subject to the terminal condition

$$V_{T+1}(s) = \begin{cases} 0 & s = 0 \\ 1 & s > 0 \end{cases}$$

7.3 Solution Algorithms

Below, we develop numerical solution algorithms for stochastic discrete time, discrete space Markov decision models. The algorithms apply to deterministic models as well, provided one views a deterministic model as a degenerate special case of the stochastic model for which the transition probabilities are all zeros or ones.

To develop solution algorithms, we must introduce some vector notation and operations. Assume that the states $S = \{1, 2, \ldots, n\}$ and actions $X = \{1, 2, \ldots, m\}$ are indexed by the first n and m integers, respectively. Let $v \in \Re^n$ denote an arbitrary value vector:

$$v_i \in \Re = \text{value in state } i;$$

and let $x \in X^n$ denote an arbitrary policy vector:

$$x_i \in X = \text{action in state } i.$$

Also, for each policy $x \in X^n$, let $f(x) \in \Re^n$ denote the n-vector of rewards earned in each state when one follows the prescribed policy:

$$f_i(x) = \text{reward in state } i, \text{ given action } x_i \text{ taken;}$$

and let $P(x) \in \Re^{n \times n}$ denote the n-by-n state transition probabilities when one follows the prescribed policy:

$$P_{ij}(x) = \text{probability of jump from state } i \text{ to } j, \text{ given action } x_i \text{ is taken.}$$

Given this notation, it is possible to express Bellman's equation for the finite horizon model succinctly as a recursive vector equation. Specifically, if $v_t \in \Re^n$ denotes the value function in period t, then

$$v_t = \max_x \{f(x) + \delta P(x) v_{t+1}\},$$

were the maximization is the vector operation induced by maximizing each row individually. Given the recursive nature of the finite horizon Bellman equation, one may compute the optimal value and policy functions v_t and x_t using backward recursion:

Algorithm: Backward Recursion

0. Initialization: Specify the rewards f, transition probabilities P, discount factor δ, terminal period T, and post-terminal value function v_{T+1}; set $t \leftarrow T$.

1. Recursion Step: Given v_{t+1}, compute v_t and x_t:

$$v_t \leftarrow \max_x \{f(x) + \delta P(x) v_{t+1}\}$$
$$x_t \leftarrow \operatorname*{argmax}_x \{f(x) + \delta P(x) v_{t+1}\}.$$

2. Termination Check: If $t = 1$, stop; otherwise set $t \leftarrow t - 1$ and return to step 1.

Each recursive step involves a finite number of matrix-vector operations, implying that the finite horizon value functions are well-defined for every period. Note however, that it may be possible to have more than one sequence of optimal policies if ties occur in Bellman's equation. Since the algorithm requires exactly T iterations, it terminates in finite time with the value functions precisely computed and at least one optimal policy obtained.

Consider now the infinite horizon Markov decision model. Given the notation above, it is also possible to express the infinite horizon Bellman equation as a vector fixed-point equation

$$v = \max_x \{f(x) + \delta P(x) v\}.$$

This vector equation may be solved using standard function iteration methods:

Algorithm: Function Iteration

0. Initialization: Specify the rewards f, transition probabilities P, discount factor δ, convergence tolerance τ, and initial guess for the value function v.

1. Function Iteration: Update the value function v:

$$v \leftarrow \max_x \{f(x) + \delta P(x)v\}.$$

2. Termination Check: If $||\Delta v|| < \tau$, set

$$x \leftarrow \operatorname{argmax}_x \{f(x) + \delta P(x)v\}$$

and stop; otherwise return to step 1.

Function iteration does not guarantee an exact solution in finitely many iterations. However, if the discount factor δ is less than one, the fixed-point map be shown to be a strong contraction. Thus, the infinite horizon value function exists and is unique, and may be computed to an arbitrary accuracy. Moreover, an explicit upper bound may be placed on the error associated with the final value function iterate. Specifically, if the algorithm terminates at iteration n, then

$$||v_n - v^*||_\infty \leq \frac{\delta}{1-\delta}||v_n - v_{n-1}||_\infty$$

where v^* is the true value function.

The Bellman vector fixed-point equation for an infinite horizon model may alternatively be recast at a rootfinding problem

$$v - \max_x \{f(x) + \delta P(x)v\} = 0$$

and solved using Newton's method. By the Envelope Theorem, the derivative of the left-hand-side with respect to v is $I - \delta P(x)$ where x is optimal for the embedded maximization problem. As such, the Newton iteration rule is

$$v \leftarrow v - (I - \delta P(x))^{-1}(v - f(x) - \delta P(x)v)$$

where P and f are evaluated at the optimal x. After algebraic simplification the update rule may be written

$$v \leftarrow (I - \delta P(x))^{-1} f(x).$$

Newton's method applied to Bellman's equation traditionally has been referred to as 'policy iteration':

Algorithm: Policy Iteration

0. Initialization: Specify the rewards f, transition probabilities P, discount factor δ, and an initial guess for v.

1. Policy Iteration: Given the current value approximant v, update the policy x:

$$x \leftarrow \underset{x}{\text{argmax}}\{f(x) + \delta P(x)v\}$$

and then update the value by setting

$$v \leftarrow (I - \delta P(x))^{-1} f(x).$$

2. Termination Check: If $\Delta v = 0$, stop; otherwise return to step 1.

At each iteration, policy iteration either finds the optimal policy or offers a strict improvement in the value function. Because the total number of states and actions is finite, the total number of admissible policies is also finite, guaranteeing that policy iteration will terminate after finitely many iterations with an exact optimal solution. Policy iteration, however, requires the solution of a linear equation system. If $P(x)$ is large and dense, the linear equation could be expensive to solve, making policy iteration slow and possibly impracticable. In these instances, the function iteration algorithm may be the better choice.

7.4 Dynamic Simulation Analysis

The optimal value and policy functions provide some insight into the nature of the controlled dynamic economic process. The optimal value function describes the benefits of being in a given state and the optimal policy function prescribes the optimal action to be taken there. However, the optimal value and policy functions provide only a partial, essentially static, picture of the controlled dynamic process. Typically, one wishes to analyze the controlled process further to learn about its dynamic behavior. Furthermore, one often wishes to know how the process is affected by changes in model parameters.

To analyze the dynamics of the controlled process, one will typically perform dynamic path and steady-state analysis. Dynamic path analysis examines how the controlled dynamic process evolves over time starting from

some initial state. Specifically, dynamic path analysis describes the path or expected path followed by the state or some other endogenous variable and how the path or expected path will vary with changes in model parameters.

Steady-state analysis examines the longrun tendencies of the controlled process over an infinite horizon, without regard to the path followed over time. Steady-state analysis of a deterministic model seeks to find the values to which the state or other endogenous variables will converge over time, and how the limiting values will vary with changes in the model parameters. Steady-state analysis of a stochastic model requires derivation of the steady-state distribution of the state or other endogenous variable. In many cases, one is satisfied to find the steady-state means and variances of these variables and their sensitivity to changes in exogenous model parameters.

The path followed by a controlled, finite horizon, deterministic, discrete, Markov decision process is easily computed. Given the state transition function g and the optimal policy functions x_t^*, the path taken by the state from an initial point s_1 can be computed as follows:

$$\begin{aligned} s_2 &= g(s_1, x_1^*(s_1)) \\ s_3 &= g(s_2, x_2^*(s_2)) \\ s_4 &= g(s_3, x_3^*(s_3)) \\ &\vdots \\ s_{T+1} &= g(s_T, x_T^*(s_T)). \end{aligned}$$

Given the path of the controlled state, it is straightforward to derive the path of actions through the relationship $x_t = x_t^*(s_t)$. Similarly, given the path taken by the controlled state and action allows one to derive the path taken by any function of the state and action.

A controlled, infinite horizon, deterministic, discrete Markov decision process can be analyzed similarly. Given the state transition function g and optimal policy function x^*, the path taken by the controlled state from an initial point s_1 can be computed from the iteration rule:

$$s_{t+1} = g(s_t, x^*(s_t)).$$

The steady-state of the controlled process can be computed by continuing to form iterates until they converge. The path and steady-state values of other endogenous variables, including the action variable, can then be computed from the path and steady-state of the controlled state.

Analysis of controlled, stochastic, discrete Markov decision processes is a bit more complicated because such processes follow a random, not a deterministic, path. Consider a finite horizon process whose optimal policy x_t^* has been derived for each period t. Under the optimal policy, the controlled state will be a finite horizon Markov chain with nonstationary transition probability matrices P_t^*, whose row i, column j element is the probability of jumping from state i in period t to state j in period $t+1$, given that the optimal policy $x_t^*(i)$ is followed in period t:

$$P_{tij}^* = \Pr(s_{t+1} = j | x_t = x_t^*(i), s_t = i)$$

The controlled state of an infinite horizon, stochastic, discrete Markov decision model with optimal policy x^* will be an infinite horizon stationary Markov chain with transition probability matrix P^* whose row i, column j element is the probability of jumping from state i in one period t to state j in the following period, given that the optimal policy $x^*(i)$ is followed:

$$P_{ij}^* = \Pr(s_{t+1} = j | x_t = x^*(i), s_t = i)$$

Given the transition probability matrix P^* for the controlled state it is possible to simulate a representative state path, or, for that matter, many representative state paths, by performing Monte Carlo simulation. To perform Monte Carlo simulation, one picks an initial state, say s_1. Having the simulated state $s_t = i$, one may simulate a jump to s_{t+1} by randomly picking a new state j with probability P_{ij}^*.

The path taken by the controlled state of an infinite horizon, stochastic, discrete Markov model may also be described probabilistically. To this end, let Q_t denote the matrix whose row i, column j entry gives the probability that the process will be in state j in period t, given that it is in state i in period 0. Then the t-period transition probability matrices Q_t are simply the matrix powers of P:

$$Q_t = P^t$$

where $Q_0 = I$. Given the t-period transition probability matrices Q_t, one can fully describe, in a probabilistic sense, the path taken by the controlled process from any initial state $s_0 = i$ by looking at the i^{th} rows of the matrices Q_t.

In most economic applications, the multiperiod transition matrices Q_t will converge to a matrix Q as t goes to infinity. In such cases, each entry

of Q will indicate the relative frequency with which the controlled decision process will visit a given state in the longrun, when starting from given initial state. In the event that all the columns of Q are identical and the longrun probability of visiting a given state is independent of initial state, then we say that the controlled state process possesses a steady-state distribution. The steady state distribution is given by the probability vector π that is the common row of the matrix Q. Given the steady-state distribution of the controlled state process, it becomes possible to compute summary measures about the longrun behavior of the controlled process, such as its longrun mean or variance. Also, it is possible to derive the longrun probability distribution of the optimal action variable or the longrun distribution of any other variables that are functions of the state and action.

7.5 Discrete Dynamic Programming Tools

In order to simplify the process of solving discrete Markov decision models, we have provided a single, unifying routine `ddpsolve` that solves such models using the dynamic programming algorithm selected by the user. The routine is executed by issuing the following command:

```
[v,x,pstar] = ddpsolve(model,alg,v)
```

Here, on input, `model` is a structured variable that contains all relevant model information, including the time horizon, the discount factor, the reward matrix, the probability transition matrix, and the terminal value function (if needed); `alg` is a string that specifies the algorithm to be used, either 'newt' for policy iteration, 'func' for function iteration, or 'back' for backward recursion; and `v` is the post-terminal value function, if the model has finite horizon, or an initial guess for the value function, if the model has infinite horizon. On output, `v` is the optimal value function, `x` is the optimal policy, and `pstar` is the optimal probability transition matrix.

The structured variable `model` contains four fields, `horizon`, `discount`, `reward`, `transition`, and `vterm` which are specified as follows:

- `horizon` - The time horizon, a positive integer or 'inf'.
- `discount` - The discount factor, positive scalar less than one.
- `reward` - An n by m matrix of rewards whose rows and columns are associated with states and columns, respectively.

- `transition` - An mn by n matrix of state transition probabilities whose rows represent this period's state and columns represent next period's state. The state transition probability matrices for the various actions are stacked vertically on top of each other, with the n by n transition probability matrix associated with action 1 at the top and the n by n transition probability matrix associated with action m at the bottom.

- `vterm` - An n by 1 vector of terminal values; is not specified if model has finite horizon; default value if not specified is zero.

The routine `ddpsolve` implements all three standard solution algorithms relying on two elementary routines. One routine takes the current value function `v`, the reward matrix `f`, the probability transition matrix `P`, and the discount factor `delta` and solves the optimization problem embedded in Bellman's equation, yielding an updated value function `v` and optimal action `x`:

```
function [v,x] = valmax(v,f,P,delta)
[m,n]=size(f);
[v,x]=max(f+delta*reshape(P*v,m,n),[],2);
```

The second routine takes a policy `x`, the reward matrix `f`, the probability transition matrix `P`, and the discount factor `delta` and returns the state reward function `fstar` and state probability transition matrix `Pstar` induced by the policy:

```
function [pstar,fstar] = valpol(x,f,P,delta)
[n,m]=size(f); i=(1:n)';
pstar = P(n*(x(i)-1)+i,:);
fstar = f(n*(x(i)-1)+i);
```

Given the `valmax` and `valpol` routines, it is straightforward to implement the backward recursion, function iteration, and policy iteration algorithms used to solve discrete Markov decision models. The Matlab script that performs backward recursion for a finite horizon model is

```
[n,m]=size(f);
x = zeros(n,T);
v = [zeros(n,T) vterm];
```

```
for t=T:-1:1
   [v(:,t),x(:,t)] = valmax(v(:,t+1),f,P,delta);
end
```

The Matlab script that performs function iteration for the infinite horizon model is

```
for it=1:maxit
   vold = v;
   [v,x] = valmax(v,f,P,delta);
   if norm(v-vold)<tol, return, end;
end
```

The Matlab script that performs policy iteration for the infinite horizon model is

```
for it=1:maxit
   vold = v;
   [v,x] = valmax(v,f,P,delta);
   [pstar,fstar] = valpol(x,f,P,delta);
   v = (eye(n,n)-delta*pstar)\fstar;
   if norm(v-vold)<tol, return, end;
end
```

The toolbox accompanying the textbook also provides two utilities for performing dynamic analysis. The first routine, `ddpsimul` is employed as follows:

```
st = ddpsimul(pstar,s1,nyrs,x)
```

On input, `pstar` is the optimal probability transition matrix induced by the optimal policy, which is generated by the routine `ddpsolve`; `x` is the optimal policy, which is also generated by the routine `ddpsolve`; `s1` is a k by 1 vector of initial states, each entry of which initiates a distinct replication of the optimized state process; and `nyrs` is the number of years for which the process will be simulated. On output, `st` is a `k` by `nyrs` vector containing `k` replications of the process, each `nyrs` in length. When the model is deterministic, the path is deterministic. When the model is stochastic, the path is generated by Monte Carlo methods. If we simulate replications all which begin from the same state, the row average of the vector `st` will provide an estimate of the expected path of the state.

The toolbox accompanying the textbook provides a second utility for performing dynamic analysis called `markov`, which is employed as follows:

```
pi=markov(pstar);
```

On input, `pstar` is the optimal probability transition matrix induced by the optimal policy, which is generated by the routine `ddpsolve`. On output, `pi` is a vector containing the invariant distribution of the optimized state process.

Finally, the toolbox accompanying the textbook provides a utility for converting the deterministic state transition rule into the equivalent degenerate probability transition matrix. The routine is employed as follows:

```
P = expandg(g);
```

On input, `g` is the deterministic state transition rule. On output, `P` is the corresponding probability transition matrix.

Given the aforementioned Matlab utilities, the most significant practical difficultly typically encountered when solving discrete Markov decision models is correctly initializing the reward and state transition matrices. We demonstrate how to implement these routines in practice in the following section.

7.6 Numerical Examples

7.6.1 Mine Management

Consider the mine management model with market price $p = 1$, initial stock of ore $\bar{s} = 100$, and annual discount factor $\delta = 0.95$.

The first step required to solve the model numerically is to specify the model parameters and to construct the state and action spaces:

```
delta = 0.9;                % discount factor
price = 1;                  % price of ore
sbar = 100;                 % initial ore stock
S = (0:sbar)';              % vector of states
n = length(S);              % number of states
X = (0:sbar)';              % vector of actions
m = length(X);              % number of actions
```

Next, one constructs the reward and transition probability matrices:

```
f = zeros(n,m);
for k=1:m
   f(:,k) = price*X(k)-(X(k)^2)./(1+S);
   f(X(k)>S,k) = -inf;
end
g = zeros(n,m);
for k=1:m
  j = max(0,S-X(k)) + 1;
  g(:,k) = j;
end
P = expandg(g);
```

Notice that a reward matrix element is set to negative infinity if the extraction level exceeds the available stock. This guarantees that the value maximization algorithm will not chose an infeasible action. Also note that we have defined the deterministic state transition rule `g` first, and then used the utility `expandg` to construct the associated probability transition matrix, which consists of mostly zeros and is stored in sparse matrix format to accelerate subsequent computations.

One then packs the essential data into the structured variable `model`:

```
model.reward     = f;
model.transition = P;
model.horizon    = inf;
model.discount   = delta;
```

Once the model data have been specified, solution of the model is relatively straightforward. To solve the infinite horizon model via policy iteration, one issues the command:

```
[vi,xi,pstari] = ddpsolve(model);
```

To solve the infinite horizon model via function iteration, one issues the command:

```
[vi,xi,pstari] = ddpsolve(model,'func');
```

Upon convergence, `vi` will be `n` vector containing the value function and `xi` will be `n` vector containing the indices of the optimal ore extractions. Note that the policy iteration algorithm was not explicitly specified because it is the default algorithm when the horizon is infinite.

To solve the model over a ten year horizon, one issues the commands

```
model.horizon = 10;
[vf,xf,pstarf] = ddpsolve(model);
```

Note that we do not have to pass the post-terminal value function, since it is identically zero, the default. Also note that the backward recursion algorithm was not explicitly specified because it is the default algorithm when the horizon is finite. Upon completion, `xf` is an `n` by `10` matrix containing the optimal ore extraction policy for all possible ore stock levels for periods 1 to `10`. The columns of `x` represent periods and its rows represent states. Similarly, `vf` is an `n` by `11` matrix containing the optimal values for all possible stock levels for periods 1 to `11`.

Once the optimal solution has been computed, one may plot the optimal value and extraction policy functions:

```
figure(1); plot(S,X(xi));
xlabel('Stock'); ylabel('Optimal Extraction');
figure(2); plot(S,vi);
xlabel('Stock'); ylabel('Optimal Value');
```

Both functions are illustrated in Figure 7.1.

To analyze the dynamics of the optimal solution, one may also plot the optimal path of the stock level over time, starting from the initial stock level, for both the finite and infinite horizon models:

```
s1 = length(S); nyrs = 10;
sipath = ddpsimul(pstari,s1,nyrs,xi);
sfpath = ddpsimul(pstarf,s1,nyrs,xf);
figure(3)
plot(1:nyrs,S(sipath),1:nyrs,S(sfpath));
legend('Infinite Horizon','Ten Year Horizon');
xlabel('Year'); ylabel('Stock');
```

As seen in Figure 7.1, one extracts the stock at a faster rate if the horizon is finite.

7.6.2 Deterministic Asset Replacement

Consider the deterministic cow replacement model with yield function $y_i = 8 + 2i - 0.25i^2$, replacement cost $c = 500$, milk price $p = 150$, and a per-cycle discount factor $\delta = 0.9$.

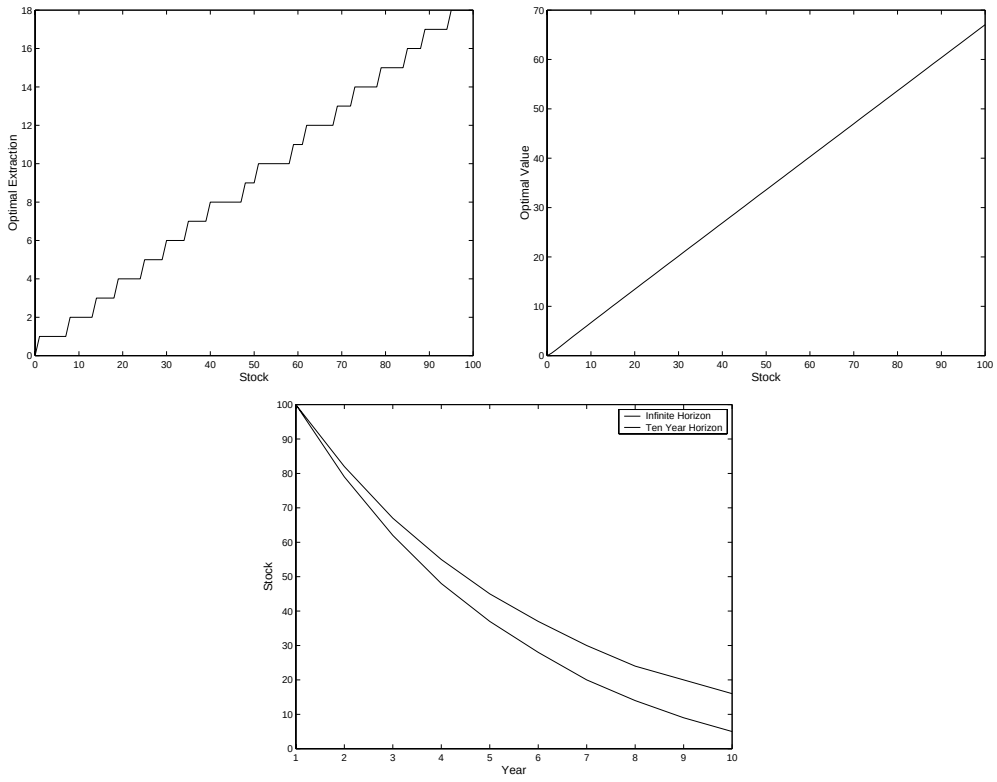

Figure 7.1: Solution to Mine Management Problem

The first step required to solve the model numerically is to specify the model parameters and to construct the state and action spaces:

```
delta = 0.9;                % discount factor
cost  = 500;                % replacement cost
price = 150;                % milk price
S = (1:10)';                % lactation states
n = length(S);              % number of states
X = ['K';'R'];              % keep or replace
m = length(X);              % number of actions
```

Next, one constructs the reward and transition probability matrices. Here, the first action is to keep the cow and the second action is to replace the cow after the current lactation:

```
y = (-0.2*S.^2+2*S+8);          % yield per lactation
f = [price*y price*y-cost];     % net revenue by action
f(10,1) = -inf;                 % replace at lactation 10
g = zeros(n,m);
for i=1:n
   g(i,1) = min(i+1,n);         % Raise number by 1, if keep
   g(i,2) = 1;                  % Number to 1, if replace
end
P = expandg(g);
```

Here, a reward matrix element is set to negative infinity for a keep decision in the tenth and final lactation because such an action is infeasible. Also note that we have defined the deterministic state transition rule `g` first, and then used the utility `expandg` to construct the associated probability transition matrix.

One then packs the essential model data into a structured variable `model`:

```
model.reward     = f;
model.transition = P;
model.horizon    = inf;
model.discount   = delta;
```

To solve the infinite horizon model via policy iteration, one issues the command:

```
[v,x] = ddpsolve(model);
```

To solve the infinite horizon model via function iteration, one issues the command:

```
[v,x] = ddpsolve(model,'func');
```

Upon convergence, `v` will be an `n` vector containing the value function and `x` will be `n` vector containing the optimal replacement decisions.

Once the optimal solution has been computed, one may plot the optimal value function:

```
figure(2); plot(s,v);
xlabel('Age'); ylabel('Optimal Value');
```

As seen in figure 7.2, the optimal policy is to replace a cow after its fifth lactation.

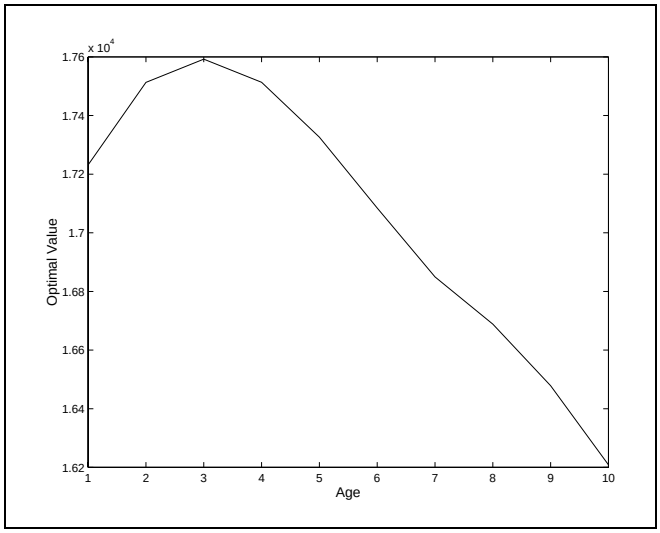

Figure 7.2

7.6.3 Stochastic Asset Replacement

Suppose now that dairy cows vary in productivity. Each cow belongs to one of 3 productivity classes, yielding 0.8, 1.0, and 1.2 times the industry baseline, respectively. Also suppose that cows from these three classes are obtained from the replacement pool with probabilities 0.2, 0.6, and 0.2, respectively.

The first step required to solve the model numerically is to specify the model parameters and to construct the state and action spaces:

```
delta = 0.9;                    % discount factor
cost  = 500;                    % replacement cost
price = 150;                    % milk price
s1 = (1:10)';                   % lactation states
s2 = [0.8;1.0;1.2];             % productivity states
n1 = length(s1);
n2 = length(s2);
[S1,S2] = cartgrid(s1,s2);      % combined state grid
n = n1*n2;                      % number of states
X = ['K','R'];                  % keep or replace
```

Note that the state space is constructed by specifying the values attainable by each state and then forming the Cartesian product using the utility

`cartgrid`.

Next, one constructs the reward matrix. Here, the first action is to keep the cow and the second action is to replace the cow after the current lactation:

```
y = (-0.2*S1.^2+2*S1+8).*S2;  % yield per lactation
f = [price*y price*y-cost];   % net revenue by action
f(S1==10,1) = -inf;           % replace at lactation 10
```

Here, a reward matrix element is set to negative infinity for a keep decision in the tenth and final lactation because such an action is infeasible.

Next, one constructs the transition probability matrix. Constructing the state transition probability matrix is a bit involved due to the multidimensional state space. Here, we set up, for each action, a four dimensional transition probability array: two dimensions for the current values of the two state variables and two dimensions for the future values of the two state variables. The four dimensional arrays are then reshaped into two-dimensional probability transition matrices and stacked for subsequent computation.

```
P1 = zeros(n1,n2,n1,n2);
P2 = zeros(n1,n2,n1,n2);
for i=1:n1
for j=1:n2
   if i<10
      P1(i,j,i+1,j) = 1;      % Up number by 1, if keep
   else
      P1(i,j,1,1) = 0.2;      % Replace after lactation 10
      P1(i,j,1,2) = 0.6;
      P1(i,j,1,3) = 0.2;
   end
   P2(i,j,1,1) = 0.2;         % Optional replacement
   P2(i,j,1,2) = 0.6;
   P2(i,j,1,3) = 0.2;
end
end
P1 = reshape(P1,n,n);
P2 = reshape(P2,n,n);
P  = sparse([P1;P2]);
```

One then packs the essential model data into a structured variable `model`:

```
model.reward     = f;
model.transition = P;
model.horizon    = inf;
model.discount   = delta;
```

To solve the infinite horizon model via policy iteration, one issues the command:

```
[v,x] = ddpsolve(model);
```

To solve the infinite horizon model via function iteration, one issues the command:

```
[v,x] = ddpsolve(model,'func');
```

Upon convergence, `v` will be an `n` vector containing the value function and `x` will be `n` vector containing the optimal replacement decisions.

Once the optimal solution has been computed, one may display the optimal replacement policy:

```
disp('Optimal Policy')
disp('     Age       Lo       Med        Hi')
fprintf('%8i %8c  %8c  %8c\n',[s1 reshape(X(x),n1,n2)]')
```

See Table 7.1.

One may also plot the optimal value function (see Figure 7.3):

```
figure(1); plot(s1,reshape(v,n1,n2))
xlabel('Age'); ylabel('Optimal Value');
legend('Low','Med','Hi')
```

To perform dynamic analysis, one first computes the stationary distribution of optimally controlled state process:

```
pi = markov(pstar);
```

Given `pi`, it is straightforward compute the average age and productivity of cows in the longrun:

```
avgage = pi'*S1;
avgpri = pi'*S2;
fprintf('\nSteady-state Age          %8.2f\n',avgage)
fprintf('\nSteady-state Productivity %8.2f\n',avgpri)
```

The invariant distribution is given in Table 7.2.

Table 7.1: Optimal Cow Replacement Policy

Age	Lo	Med	Hi
1	R	K	K
2	R	K	K
3	R	K	K
4	R	K	K
5	R	K	K
6	R	K	K
7	R	R	K
8	R	R	K
9	R	R	R
10	R	R	R

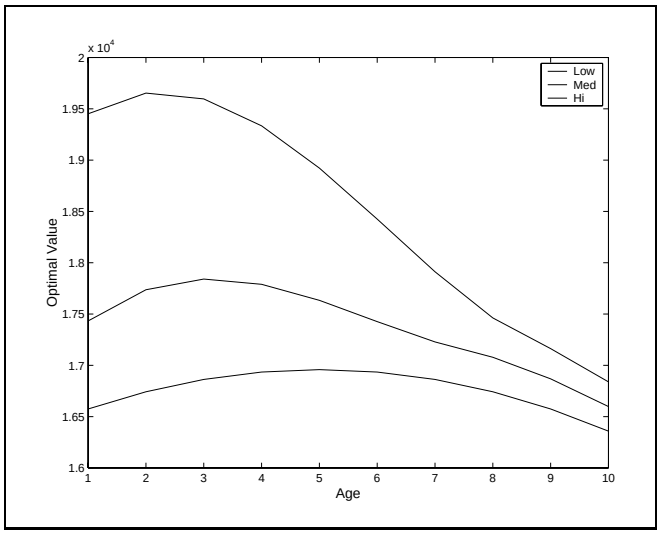

Figure 7.3

7.6.4 Option Pricing

Consider the binomial option pricing model with current asset price $p_1 = 2.00$, strike price $\bar{p} = 2.10$, annual interest rate $r = 0.05$, annual volatility $\sigma = 0.2$, and time to expiration $T = 0.5$ years that is to be divided into

Table 7.2: Stationary Distribution for Cow Replacement

Age	Lo	Med	Hi
1	0.032	0.097	0.032
2	0.000	0.097	0.032
3	0.000	0.097	0.032
4	0.000	0.097	0.032
5	0.000	0.097	0.032
6	0.000	0.097	0.032
7	0.000	0.097	0.032
8	0.000	0.000	0.032
9	0.000	0.000	0.032
10	0.000	0.000	0.000

$N = 50$ intervals.

The first step required to solve the model numerically is to specify the model parameters and to construct the state space:

```
T = 0.5;                        % years to expiration
sigma = 0.2;                    % annual volatility
r = 0.05;                       % annual interest rate
strike = 2.1;                   % option strike price
p1 = 2;                         % current asset price
N = 100;                        % number of time intervals
tau = T/N;                      % length of time intervals
delta = exp(-r*tau);            % discount factor
u = exp( sigma*sqrt(tau));      % up jump factor
q = 0.5+tau^2*(r-(sigma^2)/2)/(2*sigma); % up jump probability
price = p1*(u.^(-N:N))';        % asset prices
n = length(price);              % number of states
```

There is no need to explicitly define an action space since actions are represented by integer indices.

Next, one constructs the reward and transition probability matrices:

```
f = [ strike-price zeros(n,1) ];
P = zeros(n,n);
```

```
for i=1:n
  P(i,min(i+1,n)) = q;
  P(i,max(i-1,1)) = 1-q;
end
P = [zeros(n,n); P];
P = sparse(P);
```

Here, action 1 is identified with the exercise decision and action 2 is identified with the hold decision. Note how the transition probability matrix associated with the decision to exercise the option is identically the zero matrix. This is done to ensure that the expected future value of an exercised option always computes to zero. Also note that because the probability transition matrix contains mostly zeros, it is stored in sparse matrix format to speed up subsequent computations.

One then packs the essential model data into a structured variable `model`:

```
model.reward     = f;
model.transition = P;
model.discount   = delta;
model.horizon    = N+1;
```

To solve the finite horizon model via backward recursion, one issues the command:

```
[v,x] = ddpsolve(model);
```

Upon completion, `v(:,1)` is an `n` vector that contains the value of the American option in period 1 for different asset prices.

Once the optimal solution has been computed, one may plot the optimal value function.

```
plot(price,v(:,1)); axis([0 strike*2 -inf inf]);
xlabel('Asset Price'); ylabel('Put Option Premium');
```

This plot is given in Figure 7.4.

7.6.5 Job Search

Consider the job search model with weekly unemployment benefit $u = 55$ and psychic benefit from leisure $v = 60$. Also assume the probability of

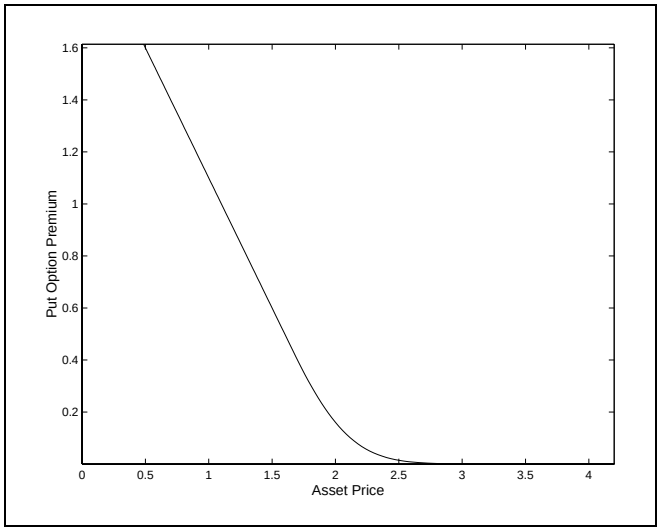

Figure 7.4

finding a job is $p = 0.90$, the probability of being fired is $q = 0.05$, and the weekly discount rate is $\delta = 0.99$. Suppose we wish to explore the optimal labor market participation policy for wages ranging from $w = 55$ to $w = 65$.

The first step required to solve the model numerically is to specify the model parameters:

```
u =  50;                    % weekly unemp. benefit
v =  60;                    % weekly value of leisure
pfind = 0.90;               % prob. of finding job
pfire = 0.10;               % prob. of being fired
delta = 0.99;               % discount factor
```

Note that by identifying both states and actions with their integer indices, one does not need to explicitly generate the state and action space.

Next, one constructs the reward and transition probability matrices. Here, we identify state 1 with unemployment and state 2 with employment, and identify action 1 with inactivity and action 2 with participation:

```
f = zeros(2,2);
f(:,1) = v;                 % gets leisure
f(1,2) = u;                 % gets benefit
```

```
P1 = sparse(zeros(2,2));
P2 = sparse(zeros(2,2));
P1(:,1) = 1;                  % remains unemployed
P2(1,1) = 1-pfind;            % finds no job
P2(1,2) = pfind;              % finds job
P2(2,1) = pfire;              % gets fired
P2(2,2) = 1-pfire;            % keeps job
P = [P1;P2];
```

One then packs the essential model data into a structured variable `model`:

```
model.reward     = f;
model.transition = P;
model.horizon    = inf;
model.discount   = delta;
```

To solve the infinite horizon model via policy iteration at different wage rates, one issues the command :

```
xtable = [];
wage=55:65;
for w=wage
   f(2,2) = w; model.reward = f; % vary wage
   [v,x] = ddpsolve(model);      % solve via policy iteration
   xtable = [xtable x];          % tabulate
end
```

Upon convergence, `xtable` will be a matrix containing the optimal labor force participation decisions at different wage rates. The table may be printed by issuing the following commands:

```
fprintf('\nOptimal Job Search Strategy')
fprintf('\n  (1=inactive, 2=active)\n')
fprintf('\nWage  Unemployed  Employed\n')
fprintf('%4i  %10i%10i\n',[wage;xtable])
```

The optimal decision rule is given in Table 7.3.

Table 7.3: Optimal Labor Participation Rule

Wage	Unemployed	Employed
55	I	I
56	I	I
57	I	I
58	I	I
59	I	I
60	I	I
61	I	A
62	A	A
63	A	A
64	A	A
65	A	A

7.6.6 Optimal Irrigation

The first step required to solve the model numerically is to specify the model parameters and to construct the state and action spaces:

```
delta =  0.9;
irrben = [-3;5;9;11];          % Irrigation Benefits to Farmers
recben = [-3;3;5;7];           % Recreational Benefits to Users
maxcap = 3;                    % maximum dam capacity
S = (0:1:maxcap)';             % vector of states
n = length(S);                 % number of states
X = (0:1:maxcap)';             % vector of actions
m = length(X);                 % number of actions
```

Next, one constructs the reward matrix:

```
f = zeros(n,m);
for i=1:n;
for k=1:m;
  if k>i
    f(i,k) = -inf;
 else
    f(i,k) = irrben(k) + recben(i-k+1);
```

```
 end
end
end
```

Here, a reward matrix element is set to negative infinity if the irrigation level exceeds the available water stock, an infeasible action.

Next, one constructs the transition probability matrix:

```
P = [];
for k=1:m
   Pk = sparse(zeros(n,n));
   for i=1:n;
      j=i-k+1; j=max(1,j); j=min(n,j);
      Pk(i,j) = Pk(i,j) + 0.4;
      j=j+1; j=max(1,j); j=min(n,j);
      Pk(i,j) = Pk(i,j) + 0.6;
   end
   P = [P;Pk];
end
```

One then packs the essential model data into a structured variable `model`:

```
model.reward     = f;
model.transition = P;
model.horizon    = inf;
model.discount   = delta;
```

To solve the infinite horizon model via policy iteration, one issues the command:

```
[v,x] = ddpsolve(model);
```

To solve the infinite horizon model via function iteration, one issues the command:

```
[v,x] = ddpsolve(model,'func');
```

Upon convergence, `v` will be `n` vector containing the value function and `x` will be `n` vector containing the optimal irrigation policy.

Once the optimal solution has been computed, one may plot the optimal value and irrigation policy functions:

```
figure(1); plot(S,X(x));
xlabel('Stock'); ylabel('Optimal Irrigation');
figure(2); plot(S,v);
xlabel('Stock'); ylabel('Optimal Value');
```

Suppose one wished to compute the steady-state stock level. One could easily do this by calling `markov` to compute the steady state distribution and integrating:

```
pi = markov(pstar);
avgstock = pi'*S;
fprintf('\nSteady-state Stock        %8.2f\n',avgstock)
```

To plot expected water level over time given that water level is currently zero, one would issue the commands

```
figure(3)
nyrs = 20;
s1=ones(10000,1);
st = ddpsimul(pstar,s1,nyrs,x);
plot(1:nyrs,mean(S(st)));
xlabel('Year'); ylabel('Expected Water Level');
```

Here, we use the function `ddpsimul` to simulate the evolution of the water level via Monte Carlo 10000 times over a 20 year horizon. The mean of the 10000 replications is then computed and plotted for each year in the simulation. The expected path, together with the optimal value and policy functions are given in Figure 7.5.

7.6.7 Bioeconomic Model

Consider the bioeconomic model with three foraging areas, predation survival probabilities $p_1 = 1$, $p_2 = 0.98$, and $p_3 = 0.90$, and foraging success probabilities $q_1 = 0$, $q_2 = 0.3$, and $q_3 = 0.8$. Also assume that successful foraging delivers $e = 4$ units of energy in all areas and that the procreation horizon is 10 periods.

The first step required to solve the model numerically is to specify the model parameters and to construct the state and action spaces:

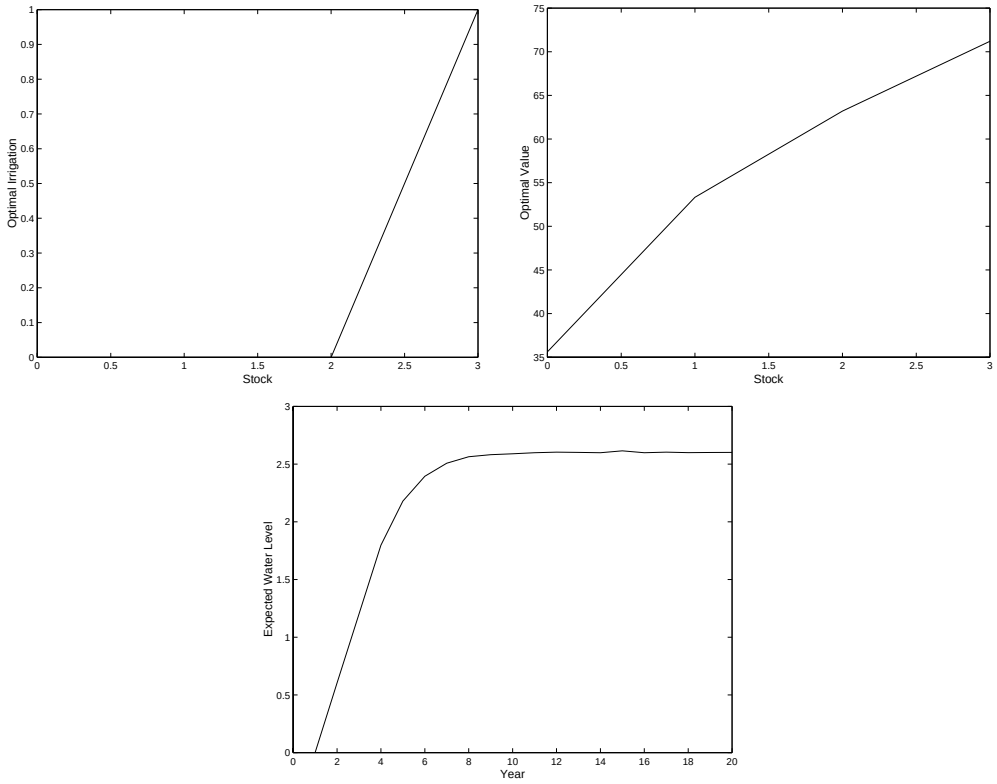

Figure 7.5: Solution to Optimal Irrigation Problem

```
T = 10;                          % foraging periods
eadd =  4;                       % energy from foraging
emax = 10;                       % energy capacity
S = 0:emax;                      % energy levels
n = length(S);                   % number of states
X = 1:3;                         % foraging areas
m = length(X);                   % number of actions
```

There is no need to explicitly define an action space since actions are represented by integer indices.

Next, one constructs the reward and transition probability matrices:

```
f = zeros(n,m);
p = [1 .98 .9];                  % predation survival prob.
```

```
q = [0 .30 .8];                % foraging success prob.
P = [];
for k=1:m
  Pk = zeros(n,n);
  Pk(1,1) = 1;
  for i=2:n;
     Pk(i,min(n,i-1+eadd)) = p(k)*q(k);
     Pk(i,i-1)             = p(k)*(1-q(k));
     Pk(i,1)               = Pk(i,1) + (1-p(k));
  end
  P = [ P ; Pk ];
end
```

Note that the reward matrix is zero because the reward is not earned until the post-terminal period. Upon the reaching the post-terminal period, either the animal is alive, earning reward of 1, or is dead, earning a reward of 0. We capture this by specifying the terminal value function as follows

```
v = ones(n,1);                 % terminal value: survive
v(1) = 0;                      % terminal value: death
```

One then packs the essential model data into a structured variable `model`:

```
model.reward     = f;
model.transition = P;
model.horizon    = inf;
model.discount   = delta;
model.vterm      = v;
```

To solve the finite horizon model via backward recursion, one issues the command:

```
[v,x] = ddpsolve(model);
```

Upon convergence, `v` will be `n` by `1` matrix containing the value function and `ix` will be `n` by `1` matrix containing the indices of the optimal foraging policy for all possible initial energy stock levels.

Once the optimal solution has been computed, one may print out the survival probabilities (see Table 7.4):

Table 7.4: Survival Probabilities

	Stock of Energy										
Period	0	1	2	3	4	5	6	7	8	9	10
1	0.00	0.59	0.71	0.80	0.82	0.83	0.85	0.92	0.93	0.93	0.93
2	0.00	0.59	0.77	0.80	0.82	0.83	0.92	0.92	0.93	0.93	1.00
3	0.00	0.64	0.77	0.80	0.82	0.91	0.92	0.92	0.93	1.00	1.00
4	0.00	0.64	0.77	0.80	0.90	0.91	0.92	0.92	1.00	1.00	1.00
5	0.00	0.64	0.77	0.88	0.90	0.91	0.92	1.00	1.00	1.00	1.00
6	0.00	0.64	0.85	0.88	0.90	0.91	1.00	1.00	1.00	1.00	1.00
7	0.00	0.72	0.85	0.88	0.90	1.00	1.00	1.00	1.00	1.00	1.00
8	0.00	0.72	0.85	0.88	1.00	1.00	1.00	1.00	1.00	1.00	1.00
9	0.00	0.72	0.85	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
10	0.00	0.72	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

```
fprintf('\nProbability of Survival\n')
disp('                          Stock of Energy')
fprintf('Period ');fprintf('%5i ',S);fprintf('\n');
for t=1:T
   fprintf('%5i ',t);fprintf('%6.2f',v(:,t)');fprintf('\n')
end
```

A similar script can be executed to print out the optimal foraging strategy (see Table 7.5).

Exercises

1. Consider a competitive price-taking firm that wishes to maximize the present value sum of current and future profits from harvesting a non-renewable resource. In year t, the firm earns revenue $p_t x_t$ where p_t is the market price for the harvested resource and x_t is the amount harvested by the firm; the firm also incurs cost αx_t^{β}, where α and β are cost function parameters. The market price takes one of two values, p_1 or p_2, according to the first-order Markov probability law:

$$\Pr[p_{t+1} = p_j | p_t = p_i] = w_{ij}.$$

Table 7.5: Optimal Foraging Strategy

	Stock of Energy										
Period	0	1	2	3	4	5	6	7	8	9	10
1	1	3	3	3	2	2	2	2	2	2	2
2	1	3	3	3	2	2	2	2	2	2	1
3	1	3	3	3	2	2	2	2	2	1	1
4	1	3	3	3	2	2	2	2	1	1	1
5	1	3	3	2	2	2	2	1	1	1	1
6	1	3	3	2	2	2	1	1	1	1	1
7	1	3	3	2	2	1	1	1	1	1	1
8	1	3	3	2	1	1	1	1	1	1	1
9	1	3	3	1	1	1	1	1	1	1	1
10	1	3	1	1	1	1	1	1	1	1	1

Assuming an annual discount factor of δ, and that harvest levels and stocks must be integers, formulate the firm's optimization problem. Specifically, formulate Bellman's functional equation, clearly identifying the state and action variables, the state and action spaces, and the reward and probability transition functions.

2. Consider a timber stand that grows by one unit of biomass per year. That is, if the stand is planted with seedlings at the beginning of year t, it will contain $t' - t$ units of biomass in year t'. Harvesting decisions are made at the beginning of each year. If the stand is harvested, new seedlings are replanted at the end of the period (so the stand has biomass 0 in the next period). The price of harvested timber is p dollars per unit and the cost of harvesting and replanting is c. The timber firm discounts the future using a discount factor of δ.

   (a) Set up the decision problem (define states, controls, reward function, transition rule).

   (b) Formulate the value function and Bellman's recursive functional equation.

   (c) For parameters values $\delta = 0.95$, $p = 1$ and $c = 5$, determine the optimal harvesting policy.

3. Suppose that a new machine costs c and that the net profit contribution of a machine of age i is p_i for $i = 0, 1, \ldots, n$, where $0 = p_n < p_{n-1} < p_{n-2} < \ldots < p_0$. Formulate the firm's profit maximization problem. Specifically, formulate Bellman's functional equation, clearly identifying the state and action variables, the state and action spaces, and the reward and probability transition functions.

4. Suppose that a new machine costs \$50 and that the net profit contribution of a machine is:

age	net profit
0	50
1	45
2	35
3	20
4+	0

   What is the optimal replacement policy for this machine.

5. Suppose that a new machine costs \$75 and that its net profit contribution in a given year

$$f(a, n) = (50 - 2.5a - 2.5a^2) * (1 - (a - n - 1)/4)$$

   depends both on its age a at the beginning of the year and number of times n that it has been serviced. At the beginning of the year, one must decide whether to keep and service the machine, keep but not service the machine, or replace the machine (one does not service a new machine). It costs \$10 to service a machine. Assuming a discount factor of 0.9, what is the optimal replacement-maintenance policy for the machine?

6. A firm operates in an uncertain profit environment. At the beginning of each period t, the firm observes its potential short-run variable profit π_t, which may be negative, and then decides whether to operate, making a short-run variable profit π_t, or to temporarily shut down, making a short-run variable profit of zero. Although the firm faces no fixed costs or shut-down costs, it incurs a start-up cost c if it reopens after a period

of inactivity. The short-run variable profit π_t follows a stationary first-order Markov process. Specifically, short-run variable profit assumes five values p_1, p_2, p_3, p_4, and p_5 with stationary transition probabilities $P_{ij} = \Pr(\pi_{t+1} = p_j | \pi_t = p_i)$.

(a) Formulate the firm's infinite horizon profit maximization problem. Specifically, formulate Bellman's functional equation, clearly identifying the state and action variables, the state and action spaces, and the reward and probability transition functions.

(b) In the standard static model of the firm, a previously open firm will shut down if its short-run variable profit p_t is negative. Is this condition sufficient in the current model?

(c) In the standard static model of the firm, a previously closed firm will reopen if its short-run variable profit p_t exceeds the start-up cost c. Is this condition necessary in the current model?

7. Consider the preceding problem under the assumption that the start-up cost is $c = 0.8$, the discount factor is $\delta = 0.95$, and the short-run variable profit assumes five values $p_1 = -1.0$, $p_2 = -0.2$, $p_3 = 0.4$, $p_4 = 1.2$, and $p_5 = 2.0$ with stationary transition probabilities:

		to				
		p_1	p_2	p_3	p_4	p_4
	p_1	0.1	0.2	0.3	0.4	0.0
	p_2	0.1	0.3	0.2	0.2	0.2
from	p_3	0.1	0.5	0.2	0.1	0.1
	p_4	0.2	0.1	0.3	0.2	0.2
	p_5	0.3	0.2	0.2	0.1	0.2.

(a) Compute the optimal operation-closure policy.

(b) What is the value of the firm?

(c) In the long-run, what percentage of the time will be firm be closed?

8. Consider the problem of optimal harvesting of a nonrenewable resource by a competitive price-taking firm:

$$\begin{array}{ll} \max & E \sum_{t=0}^{\infty} \delta^t [p_t x_t - \alpha x_t^{\beta}] \\ \text{s.t.} & s_{t+1} = s_t - x_t \end{array}$$

where $\delta = 0.9$ is the discount factor; $\alpha = 0.2$, $\beta = 1.5$, are cost function parameters; p_t is the market price; x_t is harvest; and s_t is beginning reserves. Develop a Matlab program that will solve this problem numerically assuming stock and harvest levels are integers, then answer the following questions.

(a) Graph the value function for $p = 1$ and $p = 2$.

(b) Graph the optimal decision rule for $p = 1$ and $p = 2$.

(c) Assuming an initial stocks of 100 units, graph the time path of optimal harvest for periods $t = 0$ to $t = 20$, inclusive; do so for both p=1 and p=2.

(d) Under the same assumption as in (c), graph the shadow price of stocks for periods $t = 0$ to $t = 20$. Do so both in current dollars and in year 0 dollars.

9. Consider the preceding problem, but now assume that price takes one of two values, $p = 1$ or $p = 2$ according to the following first-order Markov probability law:

$$\begin{aligned} \Pr[p_{t+1} = 1 | p_t = 1] &= 0.8 \\ \Pr[p_{t+1} = 2 | p_t = 1] &= 0.2 \\ \Pr[p_{t+1} = 1 | p_t = 2] &= 0.3 \\ \Pr[p_{t+1} = 2 | p_t = 2] &= 0.7 \end{aligned}$$

Further assume that the manager maximizes the discounted sum of expected utility over time, where utility in year t is

$$u_t = -\exp\{-\gamma(p_t x_t - \alpha x_t^{\beta})\}$$

where $\gamma = 0.2$ is the coefficient of absolute risk aversion.

(a) Write a Matlab program that solves the problem.

(b) Graph the optimal decision rule for this case and for the risk neutral case on the same graph.

(c) What is the effect of risk aversion on the rate of optimal extraction in this model?

10. Consider the article by Burt and Allison, "Farm Management Decisions with Dynamic Programming," *Journal of Farm Economics*, 45(1963):121-37. Write a program that replicates Burt and Allison's results, then compute the optimal value function and decision rule if:

    (a) the annual interest rate is 1 percent.

    (b) the annual interest rate is 10 percent.

11. Consider Burt and Allison's farm management problem. Assume now that the government will subsidize fallow land at \$25 per acre, raising the expected return on a fallow acre from a \$2.33 loss to a \$22.67 profit. Further assume, as Burt and Allison implicitly have, that cost, price, yield, and return are determinate at each moisture level:

    (a) Compute the optimal value function and decision rule.

    (b) Derive the steady-state distribution of the soil moisture level under the optimal policy.

    (c) Derive the steady-state distribution of return per acre under the optimal policy.

    (d) Derive the steady-state mean and variance of return per acre under the optimal policy.

12. At the beginning of every year, a firm must decide how much to produce over the coming year in order to meet the demand for its product. The demand over any year is known at the beginning of the year, but varies annually, assuming serially independent values of 5, 6, 7, or 8 thousand units with probabilities 0.1, 0.3, 0.4, and 0.2, respectively. The firm's cost of production in year t is $10q_t + (q_t - q_{t-1})^2$ thousand dollars, where q_t is thousands of units produced in year t. The product sells for \$20 per unit and excess production can either be carried over to the following year at a cost of \$2 per unit or disposed of for free. The firm's production and storage capacities are 8 thousand and 5 thousand units per annum, respectively. The annual discount factor is 0.9. Assuming that the firm meets its annual demand exactly, and that production and storage levels must be integer multiples of one thousand units, answer the following questions:

    (a) Under what conditions would the firm use all of its storage capacity?

(b) What is the value of firm and what is its optimal production if its previous year's production was 5 thousand units, its carryin is 2 thousand units, and the demand for the coming year is 7 units?

(c) What would be the production levels over the subsequent three years if the realized demands were 6, 5, and 8 units, respectively?

Chapter 8

Discrete Time Continuous State Dynamic Models: Theory

We now turn our attention to discrete time dynamic economic models with state variables that may assume a continuum of values. Three classes of discrete time, continuous state dynamic economic models are examined. One class includes models of centralized decisionmaking by individuals, firms, or institutions. Examples of continuous state decision models admitting a continuum of choices include a central planner managing the harvest of a natural resource so as to maximize social welfare, an entrepreneur planning production and investment so as to maximize the present value of her firm, a consumer making consumption and savings decisions so as to maximize his expected lifetime utility. Examples of continuous state dynamic decision models requiring dichotomous or binary choices include a financial investor deciding when to exercise a put option, a capitalist deciding whether to enter or exit an industry, and a producer deciding whether to keep or replace a physical asset.

A second class of discrete time continuous state dynamic model examined includes models of strategic gaming among a small number of individuals, firms, or institutions. Dynamic game models attempt to capture the behavior of a small group of dynamically optimizing agents when the policy pursued by one agent directly affects the welfare of another. Examples include a two national grain marketing boards deciding quantities of grain to sell on world markets and two individuals deciding how much to work and invest in the presence of an income risk-sharing arrangement.

A third class of discrete time continuous state dynamic economic model

examined includes partial and general equilibrium models of collective, decentralized economic behavior. Dynamic equilibrium models characterize the behavior of a market, economic sector, or entire economy through intertemporal arbitrage conditions that are enforced by the collective action of atomistic dynamically optimizing agents. Often the behavior of agents at a given date depends on their expectations of what will happen at a future date. If it is assumed that agent's expectations are consistent with the implications of the model as a whole, then agents are said to possess rational expectations. Examples of rational expectations models include arbitrage pricing models for financial assets and physical commodities.

Dynamic optimization and equilibrium models are closely related. The solution to a continuous state dynamic optimization may often be equivalently characterized by first-order intertemporal equilibrium conditions obtained by differentiating Bellman's equation. Conversely, many dynamic equilibrium problems can be "integrated" into equivalent optimization formulations. Whether cast in optimization or equilibrium form, most discrete time continuous state dynamic economic models pose infinite-dimensional fixed-point problems that lack closed-form solution. This chapter provides an introduction to the theory of discrete time continuous state dynamic economic models. The subsequent chapter is devoted to numerical methods that may be used to solve and analyze such models.

8.1 Continuous State Dynamic Programming

A discrete time, continuous state Markov decision model involves a reward function that depends on a state variable that may assume any one of an infinite number of values contained in a closed convex set. Such models may be classified according to whether the action space is also a continuum or whether it is a finite set. We treat these two classes of decision models separately, given that they may be analyzed and solved numerically in ways that are similar in some respects, but dissimilar in other respects. When discussing continuous state, discrete action Markov decision models, we limit our attention to models with binary choices, without significant loss of generality.

The discrete time, continuous state and action Markov decision model has the following structure: In every period t, an agent observes the state of an economic process s_t, takes an action x_t, and earns a reward $f(s_t, x_t)$ that

depends on both the state of the process and the action taken. The state space $S \in \Re^n$, which contains all the states attainable by the process, and the action space $X \in \Re^m$, which contains all actions that may be taken by the agent, are both closed convex sets. The state of the economic process follows a controlled Markov probability law. Specifically, the state of the economic process in period $t+1$ will depend on the state and action in period t and an exogenous random shock ϵ_{t+1} that is unknown in period t:

$$s_{t+1} = g_t(s_t, x_t, \epsilon_{t+1}).$$

The agent seeks a policy $\{x_t^*\}_{t=1}^T$ of state-contingent actions $x_t = x_t^*(s_t)$ that will maximize the present value of current and expected future rewards, discounted at a per-period factor δ:

$$E \sum_t \delta^t f(s_t, x_t).$$

In the continuous state and action Markov decision model, the exogenous random shocks ϵ_t are assumed identically distributed over time, mutually independent, and independent of past states and actions. The reward functions f and the state transition functions g are assumed to be twice continuously differentiable on S and X and the per-period discount factor δ is assumed to be less than one. In some instances, the set of actions available to the agent may vary with the state of the process s. In such cases, the restricted action space is denoted $X(s)$. Continuous state, continuous action Markov decision models may further be classified according to whether their horizon is finite or infinite and whether they are stochastic or deterministic.

Like the discrete Markov decision problem, the discrete time continuous state continuous action Markov decision problem may be analyzed using dynamic programming methods based on Bellman's Principle of Optimality. The Principle of Optimality applied to the discrete time continuous state continuous action Markov decision model yields Bellman's recursive functional equation:

$$V_t(s) = \max_{x \in X(s)} \{f(s,x) + \delta E_\epsilon V_{t+1}(g(s,x,\epsilon))\}, \qquad s \in S.$$

Here, the value function $V_t(s)$ gives the maximum attainable sum of current and expected future rewards, given that the process is in state s in t.

In a finite horizon model, we adopt the convention that the optimizing agent faces decisions up to and including a final decision period $T < \infty$. The

agent faces no decisions after the terminal period T, but may earn a final reward V_{T+1} in period $T+1$. In many applications, the post-terminal value function V_{T+1} is identically zero, indicating that no rewards are earned by the agent beyond the terminal decision period. In other applications, V_{T+1} may specify a salvage value earned by the agent after making his final decision in period T. Given the post-terminal value function, the finite horizon discrete time continuous state continuous action Markov decision model may be solved recursively, at least in principle, by repeated application of Bellman's equation: Having V_{T+1}, solve for $V_T(s)$ for all states s; having V_T, solve for $V_{T-1}(s)$ for all states s; having V_{T-1}, solve for $V_{T-2}(s)$ for all states s; and so on, until $V_0(s)$ is derived for all states s.

The value function of the infinite horizon discrete time continuous state continuous action Markov decision model will the same for every period and thus may be denoted simply by V. The infinite horizon value function V is characterized as the solution to the Bellman functional fixed-point equation

$$V(s) = \max_{x \in X(s)} \{f(s,x) + \delta E_\epsilon V(g(s,x,\epsilon))\}, \qquad s \in S.$$

If the discount factor δ is less than one and the reward function f is bounded, the mapping underlying Bellman's equation is a strong contraction on the space of bounded continuous functions and, thus, by The Contraction Mapping Theorem, will possess an unique solution.

The simplest continuous state binary choice Markov decision model is the optimal stopping problem. At each point in time t the agent is offered a one-time reward $f(s_t)$ that depends on the state of some purely exogenous stochastic economic process s_t. The agent must then decide whether to accept the offer, receiving the reward, or decline the offer, forgoing the reward and waiting another period for a hopefully better reward. The underlying economic process s_t is a continuous state Markov process with transition function $s_{t+1} = g(s_t, \epsilon_{t+1})$, where the ϵ_t are an i.i.d. and serially independent. The Bellman equation of the infinite horizon optimal stopping model takes the simple form

$$V(s) = \max\{f(s), \delta E_\epsilon V(g(s,\epsilon))\}, \qquad s \in S.$$

The value function V specifies the present value of the expected maximum reward, given that the exogenous process is in state s.

Another important continuous state binary choice Markov decision model is the optimal switching problem. In the optimal switching model, an agent

must decide whether to be dormant (0) or active (1) in the current period. The reward $f(s_t, i_t, j_t)$ earned by the agent in period t depends on the agent's activity in the preceding period i_t, the agent's activity in the current period j_t, and a continuous-valued, purely exogenous state variable s_t governed by the Markov transition law $s_{t+1} = g_t(s_t, \epsilon_{t+1})$. The Bellman equation of the infinite horizon optimal switching model takes the form

$$V(s,i) = \max_{j=0,1}\{f(s,i,j) + \delta E_\epsilon V(g(s,\epsilon),j)\}, \qquad s \in S, i = 0, 1.$$

The value function V specifies the maximum attainable sum of current and expected future rewards, given that the exogenous process is in state s and the agent's state of activity in the preceding period was i.

Yet another important continuous state binary choice Markov decision model is the optimal asset replacement model. In the optimal asset replacement model, an agent must decide when to incur the cost of replacing an aging asset with a newer, potentially more productive asset. The reward earned by the agent in any period equals the earnings generated by the asset used in that period less replacement costs, if any. Specifically, if the agent begins period t with an asset of age a_t, he receives a reward $f(s_t, a_t)$, if he keeps the asset, and a reward $f(s_t, 0) - c$, if he replaces it. The earnings generated by an asset depend not only on the age of the asset, but also a continuous-valued, purely exogenous state variable s_t governed by the Markov transition law $s_{t+1} = g_t(s_t, \epsilon_{t+1})$. Assuming that the asset becomes totally unproductive at age $\bar{a}$, the Bellman equation of the infinite horizon optimal asset replacement model takes the form

$$V(s,a) = \max\{f(s,a) + \delta E_\epsilon V(g(s,\epsilon), a+1), f(s,0) - c + \delta E_\epsilon V(g(s,\epsilon),1)\}$$

for $s \in S$ and $a = 0, 1, \ldots, \bar{a}$, provided we agree to interpret $a + 1$ to mean $\bar{a}$ when $a = \bar{a}$. The value function V specifies the maximum attainable sum of current and expected future rewards, given that the exogenous process is in state s and the age of the asset at the beginning of the period is a.

Although both finite- and infinite-horizon Bellman equations involving continuous states are guaranteed to have solutions if the reward function is bounded, they will often lack closed-form solution. The problem lies with the continuous state space, which contains an infinite number of points. Except in rare special cases, it is not possible to derive analytically an explicit closed form expression for the period t value function of the finite horizon model,

even if the period $t+1$ value function is known and possesses a closed-form. In these instances, solving Bellman's equation requires explicitly solving an infinite number of optimization problems, one for each state. This is an impracticable task. Except in a very small number of special cases, one can only solve a discrete time continuous state Bellman equation numerically, a matter that we take up the following chapter.

8.2 Euler Equilibrium Conditions

Like many optimization problems, the solution to Markov decision models with continuous state and action spaces can be characterized by "first-order" equilibrium conditions. Characterizing the solution to a continuos state and action Markov decision problem through its equilibrium conditions, widely called the Euler conditions, serves two purposes. First, the Euler conditions admit an intertemporal arbitrage interpretation that help the analyst understand and explain the essential features of the optimized dynamic economic process. Second, the Euler conditions can, in many instances, be solved more easily than Bellman's equation for the optimal solution of the Markov decision model. Below, we derive the Euler conditions for the infinite horizon model, leaving the derivation of the Euler conditions for the finite horizon model as an exercise for the reader.

The equilibrium conditions of the continuous state and action Markov decision problem involve, not the value function, but its derivative

$$\lambda(s) \equiv V'(s).$$

We call λ the shadow price function. It represents the value of the marginal unit of state variable to the optimizer or, equivalently, the price that the optimizer imputes to the state variable.

Assume that both the state and action spaces are closed convex nonempty sets and that the reward functions f and the state transition functions g are continuously differentiable of all orders. The equilibrium conditions for discrete time continuous state continuous choice Markov decision problem are derived by applying the Karush-Kuhn-Tucker and Envelope Theorems to the optimization problem embedded in Bellman's equation. Assuming actions are unconstrained, the Karush-Kuhn-Tucker conditions for the embedded unconstrained optimization problem imply that the optimal action x, given

state s in period t, satisfies the equimarginality condition:

$$f_x(s,x) + \delta E_\epsilon \left[\lambda(g(s,x,\epsilon))g_x(s,x,\epsilon)\right] = 0.$$

The Envelope Theorem applied to the same problem implies:

$$f_s(s,x) + \delta E_\epsilon \left[\lambda(g(s,x,\epsilon)) \cdot g_s(s,x,\epsilon)\right] = \lambda(s).$$

Here, f_x, f_s, g_x, and g_s denote partial derivatives.

In certain applications, the transition function is independent of s so that $g_s = 0$. In these instances, it is possible to substitute the expression derived using the Envelope theorem into the expression derived using the Karush-Kuhn-Tucker condition. This allows us to eliminate the shadow price function as an unknown, and simplify the Euler conditions into a single functional equation in a singe unknown, the optimal policy function x:

$$f_x(s,x) + \delta E_\epsilon \left[f_s(g(s,x,\epsilon),x)g_x(s,x,\epsilon)\right] = 0.$$

This equation, when it exists, is known as the Euler equation.

The Euler conditions take a different form when actions are subject to constraints. Suppose, for example, that feasible actions are subject to bounds of the form

$$X(s) = \{x \mid a(s) \leq x \leq b(s)\},$$

where a and b are differentiable functions of the state s. In these instances, the Euler conditions take the form of a functional complementarity problem:

$$a(s) \leq x \leq b(s)$$

$$x > a(s) \Longrightarrow f_x(s,x) + \delta E_\epsilon \left[\lambda(g(s,x,\epsilon)) \cdot g_x(s,x,\epsilon)\right] \geq 0$$

$$x < b(s) \Longrightarrow f_x(s,x) + \delta E_\epsilon \left[\lambda(g(s,x,\epsilon)) \cdot g_x(s,x,\epsilon)\right] \leq 0$$

$$f_s(s,x) + \delta E_\epsilon \left[\lambda(g(s,x,\epsilon)) \cdot g_s(s,x,\epsilon)\right] + \mu^- a'(s) + \mu^+ b'(s) = \lambda(s).$$

where

$$\begin{aligned} \mu^+ &= \max(0, f_x(s,x) + \delta E_\epsilon \left[\lambda(g(s,x,\epsilon)) \cdot g_x(s,x,\epsilon)\right]) \\ \mu^- &= \min(0, f_x(s,x) + \delta E_\epsilon \left[\lambda(g(s,x,\epsilon)) \cdot g_x(s,x,\epsilon)\right]) \end{aligned}$$

are the shadow prices of the upper and lower bounds, respectively.

An analyst is often interested with the longrun tendencies of the optimized process. If the model is deterministic, it may possess a well-defined steady-state to which the process will converge over time. The steady-state is characterized by the solution to a nonlinear equation. More specifically, the steady-state of an unconstrained deterministic problem, if it exists, consists of a state s^*, an action x^*, and shadow price λ^* such that

$$f_x(s^*, x^*) + \delta\lambda^* g_x(s^*, x^*) = 0$$

$$\lambda^* = f_s(s^*, x^*) + \delta\lambda^* g_s(s^*, x^*)$$

$$s^* = g(s^*, x^*).$$

The steady-state of a constrained deterministic dynamic optimization problem can be similarly stated, except that it takes the form of a nonlinear complementarity problem, rather than a nonlinear equation.

Knowledge of the steady-state of a deterministic Markov decision problem is often very useful. For most well-posed deterministic problems, the optimized process will converge to the steady-state, regardless of initial condition. The steady-state therefore unequivocally characterizes the longrun behavior of the process. The analyst, moreover, will often be satisfied to understand the dynamics of the process around the steady-state, given that this is the region where the process tends to reside. The steady-state conditions are equations or complementarity conditions that can be analyzed algebraically. In particular, the derivative of the longrun value of an endogenous variable with respect to model parameters can often be derived using standard differential calculus, even if the dynamic model itself lacks a closed-form solution.

If the discrete time continuous state model is stochastic, the model will not converge to a specific state and action and the longrun behavior of the model can only be described probabilistically. In these cases, however, it is often practically useful to derive the steady-state of the deterministic "certainty-equivalent" problem obtained by fixing all exogenous random shocks at their respective means. Knowledge of the certainty-equivalent steady-state can assists the analyst by providing a reasonable initial guess for the optimal policy, value, and shadow price functions in iterative numerical solution algorithms. Also, one can often solve a hard stochastic dynamic model by first solving the certainty-equivalent model, and then solving a series of models obtained by gradually perturbing the variance of the shock

from zero back to its true level, always using the solution of one model as the starting point for the algorithm used to solve the subsequent model.

8.3 Linear-Quadratic Control

Before proceeding to more complicated continuous state Markov decision models we discuss a special case: the linear-quadratic control model. The linear-quadratic control problem is a Markov decision model with a quadratic reward function

$$f(s,x) = F_0 + F_s s + F_x x + 0.5 s' F_{ss} s + s' F_{sx} x + 0.5 x' F_{xx} x$$

and a linear state transition function with additive shock

$$g(s,x,\epsilon) = G_0 + G_s s + G_x x + \epsilon.$$

Here, s is an n-by-1 state vector, x is an m-by-1 action vector, F_0 is a known constant, F_s is a known 1-by-n vector, F_x is a known 1-by-m vector, F_{ss} is a known n-by-n matrix, F_{sx} is a known n-by-m matrix, F_{xx} is a known m-by-m matrix, G_0 is a known n-by-1 vector, G_s is a known n-by-n matrix, and G_x is a known n-by-m vector. Without loss of generality, the shock ϵ is assumed to have a mean of zero. The linear-quadratic control problem admits no constraints on the action.

The linear-quadratic is of special importance because it is one of the few discrete time continuous state Markov decision models with known analytic solution. By a conceptually simple but algebraically burdensome induction proof omitted here, one can show that the solution to the infinite horizon linear-quadratic control model takes a particularly simple form. Specifically, both the optimal policy and shadow price functions are linear in the state variable:

$$x(s) = X_0 + X_s s$$

$$\lambda(s) = \Lambda_0 + \Lambda_s s.$$

Here, X_0 is an m-by-1 vector, X_s is an m-by-n matrix, Λ_0 is an n-by-1 vector, and Λ_s is an n-by-n matrix.

The parameters Λ_0 and Λ_s of the shadow price function are characterized by the nonlinear vector fixed point Riccati equations

$$\begin{aligned}\Lambda_0 &= -[\delta G_s' \Lambda_s G_x + F_{sx}][\delta G_x' \Lambda_s G_x + F_{xx}']^{-1}[\delta G_x'[\Lambda_s G_0 + \Lambda_0] + F_x] \\ &\quad +\delta G_s'[\Lambda_s G_0 + \Lambda_0] + F_s \\ \Lambda_s &= -[\delta G_s' \Lambda_s G_x + F_{sx}][\delta G_x' \Lambda_s G_x + F_{xx}']^{-1}[\delta G_x' \Lambda_s G_s + F_{sx}'] \\ &\quad +\delta G_s' \Lambda_s G_s + F_{ss}.\end{aligned}$$

These finite dimensional fixed-point equations can typically be solved in practice using a simple function iteration scheme applied to both equations. Alternatively, the recursive structure of these equations allow one to first solve for Λ_s by applying function iteration to the second equation, and then solve for Λ_0 by applying function iteration to the first equation. Once the parameters of the shadow price function have been computed, one can easily compute the parameters of the optimal policy:

$$X_0 = -[\delta G_x' \Lambda_s G_x + F_{xx}']^{-1}[\delta G_x'[\Lambda_s G_0 + \Lambda_0] + F_x]$$

$$X_s = -[\delta G_x' \Lambda_s G_x + F_{xx}']^{-1}[\delta G_x' \Lambda_s G_s + F_{sx}']$$

The relative simplicity of the linear-quadratic control problem derives from the fact that the optimal policy and shadow price functions are known to be linear, and thus belong to a finite dimensional family. The parameters of the linear functions, moreover, are characterized as the solution to a well-defined nonlinear vector fixed-point equation. Thus, the apparently infinite-dimensional Euler functional fixed-point equation may be converted into finite-dimensional vector fixed-point equation and solved using standard nonlinear equation solution methods. This simplification, unfortunately, is not generally possible for other types of discrete time continuous state Markov decision models.

A second simplifying feature of the linear-quadratic control problem is that the shadow price and optimal policy functions do not depend on the distribution of the state shock. This is known as the certainty-equivalence property of the linear-quadratic control problem. It asserts that the solution of the stochastic problem is the same as the solution of the deterministic problem obtained by fixing the state shock ϵ at its mean of zero. Certainty equivalence also is not a property of more general discrete time continuous state Markov decision models.

Because linear-quadratic control models are relatively easy to solve, many analysts compute approximate solutions to more general Markov decision models using the method of linear-quadratic approximation. Linear quadratic approximation calls for all constraints of the general problem to be discarded and for its reward and transition functions to be replaced with by their second- and first-order approximations about the steady-state. This approximation method, which is illustrated in the following chapter, works well in some instances, for example, if the state transition rule is linear, constraints are nonbinding and non existent and if the shocks have small variation. However, in most Economic applications, linear-quadratic approximation will often render highly inaccurate solutions that differ not only quantitatively but also qualitatively from the true solution. For this reason, we strongly discourage the use of linear-quadratic approximation, except in those cases where the assumptions of the linear quadratic model are know to hold globally for the model under consideration.

8.4 Economic Examples

8.4.1 Asset Replacement

Suppose that a new machine costs K and that the output of a machine of age a is $q = q(a)$, where q initially increases in a, but eventually declines, reaching zero at a machine obsolescence age $\bar{a}$. Also suppose that the price p obtained per unit of output is a purely exogenous log-normal random process governed by the transition rule

$$\log(p_{t+1}) = \alpha + \beta(\log p_t - \alpha) + \epsilon_{t+1}$$

where the ϵ_t are serially independent and identically normally distributed with mean 0 and standard deviation σ. What is the value of the firm and what is the optimal replacement policy?

This is an infinite horizon, stochastic model with time $t \in \{0, 1, 2, \ldots\}$ measured in periods. The model has a continuous state variable

$$\begin{aligned} p_t &= \text{output price in period } t \\ p_t &\in (0, \infty) \end{aligned}$$

and a discrete state variable

$$a_t = \text{asset age at beginning of period } t$$

$$a_t \in \{0, 1, 2, \ldots, \bar{a}\}.$$

The choice variable i_t is dichotomous: either keep (0) or replace (1) the asset.

The reward earned by the optimizing agent is

$$p_t q(a_t) - c(i_t) = \text{net revenue in } t$$

where $c(0) = 0$ and $c(1) = K$. And the continuous state variable transitions are governed by

$$p_{t+1} = g(p_t, \epsilon_{t+1}) = \exp\{\alpha + \beta(\log p_t - \alpha) + \epsilon_{t+1}\}.$$

The value function, which measures the value of having an asset of age a, satisfies Bellman's equation

$$V(p, a) = \max\{pq(a) + \delta E_\epsilon V(g(p, \epsilon), a + 1), pq(0) - K + \delta E_\epsilon V(g(p, \epsilon), 1)\}$$

8.4.2 Industry Entry and Exit

A firm operates in an uncertain profit environment. At the beginning of each period t, the firm observes its potential short-run variable profit π_t, which may be negative, and then decides whether to operate, making a short-run variable profit π_t, or to not operate, making a short-run variable profit of zero. Although the firm faces no fixed costs or shut-down costs, it incurs a start-up cost K if it reopens after a period of inactivity. The short-run variable profit π_t follows a stationary first-order Markov process

$$\pi_{t+1} = g(\pi, \epsilon_{t+1}) = \bar{\pi} + \beta(\pi_t - \bar{\pi}) + \epsilon_{t+1}$$

where the ϵ_t are serially independent and identically normally distributed with mean 0 and standard deviation σ. What is the value of the firm and what is the optimal entry-exit policy?

This is an infinite horizon, stochastic model with time $t \in \{0, 1, 2, \ldots\}$ measured in periods. The model has a continuous state variable

$$\begin{aligned} \pi_t &= \text{short run profit potential, period } t \\ \pi_t &\in (-\infty, \infty) \end{aligned}$$

The choice variable j_t is dichotomous: either operate (1) or do not operate (0) in period t.

The reward earned by the optimizing agent is

$$\pi_t j_t - c(i_t, j_t) = \text{net revenue in } t$$

where $c(0,1) = K$ but is zero otherwise.

The value function, which measures the value of the firm, satisfies Bellman's equation

$$V(\pi, i) = \max_{j=0,1}\{\pi j - c(i,j) + \delta E_\epsilon V(g(\pi, \epsilon), j)\}$$

8.4.3 Option Pricing

An American put option gives the holder the right, but not the obligation, to sell a specified quantity of a commodity at a specified strike price on or before a specified expiration date. In the discrete-time Black-Scholes option pricing model, the price of the commodity is assumed to follow a purely exogenous log-normal random process governed by the transition rule

$$\log(p_{t+1}) = \alpha + \beta(\log p_t - \alpha) + \epsilon_{t+1}$$

where the ϵ_t are serially independent and identically normally distributed with mean 0 and standard deviation σ. Assuming the current price of the commodity is p_0, what is the value of an American put option if it has a strike price $\bar{p}$ and expires T periods from today?

This is a finite horizon, stochastic model with time $t \in \{0, 1, 2, \ldots, T\}$ measured in periods. The model has a continuous state variable

$$\begin{aligned} p_t &= \text{commodity price in period } t \\ p_t &\in (0, \infty). \end{aligned}$$

The choice variable j_t is dichotomous: either hold (0) or exercise (1) the asset.

The reward earned by the optimizing agent is

$$\bar{p} - p_t = \text{exercise value in period } t$$

if the option is exercised, but is zero otherwise. And the state variable transitions are governed by

$$p_{t+1} = g(p_t, \epsilon_{t+1}) = \exp\{\alpha + \beta(\log p_t - \alpha) + \epsilon_{t+1}\}.$$

The value function, which measures the value of an unexercised option, satisfies Bellman's equation

$$V(p) = \max\{p - \bar{p}, \delta E_\epsilon V(g(p, \epsilon))\}$$

8.4.4 Optimal Growth

Consider an economy comprising a single composite good. Each year t begins with a predetermined amount of the good s_t, of which an amount x_t is invested and the remainder is consumed. The social welfare derived from consumption in year t is $u(s_t - x_t)$. The amount of good available in year $t+1$ is $s_{t+1} = \gamma x_t + \epsilon_{t+1} f(x_t)$ where γ is the capital survival rate, f is the aggregate production function, and ϵ_{t+1} is a positive production shock with mean 1. What consumption-investment policy maximizes the sum of current and expected future welfare over an infinite horizon?

This is an infinite horizon, stochastic model with time $t \in \{0, 1, 2, \ldots\}$ measured in years. The model has a single state variable

$$\begin{aligned} s_t &= \text{stock of good at beginning of year } t \\ s_t &\in [0, \infty) \end{aligned}$$

and a single action variable

$$x_t = \text{amount of good invested in year } t$$

subject to the constraint

$$0 \leq x_t \leq s_t.$$

The reward earned by the optimizing agent is

$$u(s_t - x_t) = \text{social utility in } t.$$

State transitions are governed by

$$s_{t+1} = \gamma x_t + \epsilon_{t+1} f(x_t)$$

where

$$\epsilon_t = \text{productivity shock in year } t.$$

The value function, which gives the sum of current and expected future social welfare, satisfies Bellman's equation

$$V(s) = \max_{0 \leq x \leq s} \{u(s - x) + \delta E V(\gamma x + \epsilon f(x))\}, \qquad s > 0.$$

Assuming $u'(0) = -\infty$ and $f(0) = 0$, the solution to Bellman's equation will always be internal, and the shadow price function, which gives the shadow price of stock, satisfies the Euler equilibrium conditions:

$$u'(s-x) - \delta E\left[\lambda(\gamma x + \epsilon f(x)) \cdot (\gamma + \epsilon f'(x))\right] = 0$$

$$\lambda(s) = u'(s-x).$$

Thus, along the optimal path,

$$u'_t = \delta E_t\left[u'_{t+1} \cdot (\gamma + \epsilon_{t+1} f'_t)\right]$$

where u'_t is marginal utility and $\epsilon_{t+1} f'_t$ is the ex-post marginal product of capital. That is, on the margin, the utility derived from a unit of good today must equal the discounted expected utility derived from investing the good and consuming it and its product tomorrow.

The certainty-equivalent steady-state is obtained by fixing ϵ at its mean 1. The certainty-equivalent steady-state stock of good s^*, investment level x^*, and shadow price λ^* are characterized by the nonlinear equation system

$$u'(s^* - x^*) = \delta\lambda^*(\gamma + f'(x^*))$$

$$\lambda^* = u'(s^* - x^*)$$

$$s^* = \gamma x^* + f(x^*).$$

The certainty-equivalent steady-state conditions imply the golden rule: $1 - \gamma + r = f'(x^*)$. That is, in deterministic steady-state, the marginal product of capital equals the capital depreciation rate plus the interest rate. Totally differentiating the equation system above with respect to the interest rate r:

$$\frac{\partial s^*}{\partial r} = \frac{1+r}{f''} < 0$$

$$\frac{\partial x^*}{\partial r} = \frac{1}{f''} < 0$$

$$\frac{\partial \lambda^*}{\partial r} = \frac{u'' r}{f''} > 0.$$

That is, a permanent rise in the interest rate will reduce the deterministic steady-state supply and investment, and will raise the shadow price.

8.4.5 Renewable Resource Problem

A social planner wishes to maximize the discounted sum of net social surplus from harvesting a renewable resource over an infinite horizon. For year t, let s_t denote the resource stock at the beginning of the year, let x_t denote the amount of the resource harvested, let $c_t = c(x_t)$ denote the total cost of harvesting, and let $p_t = p(x_t)$ denote the market clearing price. Growth in the stock level is given by $s_{t+1} = g(s_t - x_t)$. What is the socially optimal harvest policy?

This is an infinite horizon, deterministic model with time $t \in \{0, 1, 2, \ldots\}$ measured in years. There is one state variable,

$$\begin{aligned} s_t &= \text{stock of resource at beginning of year } t \\ s_t &\in [0, \infty), \end{aligned}$$

and one action variable,

$$x_t = \text{amount of resource harvested in year } t,$$

subject to the constraint

$$0 \leq x_t \leq s_t.$$

The reward earned by the optimizing agent is

$$\int_0^{x_t} p(\xi)\, d\xi - c(x_t).$$

State transitions are governed by

$$s_{t+1} = g(s_t - x_t).$$

The value function, which gives the net social value of resource stock, satisfies Bellman's equation

$$V(s) = \max_{0 \leq x \leq s} \{ \int_0^x p(\xi)\, d\xi - c(x) + \delta V(g(s - x)) \}.$$

Assuming $p(0) = \infty$ and $g(0) = 0$, the solution to the optimization problem embedded in Bellman's equation will be internal. Under these assumptions the shadow price function satisfies the Euler conditions, which stipulate that for every stock level $s > 0$ there is a harvest level x such that

$$p(x) = c'(x) + \delta \lambda(g(s - x)) g'(s - x)$$

$$\lambda(s) = \delta\lambda(g(s-x))g'(s-x).$$

Thus, along the optimal path

$$p_t = c'_t + \lambda_t$$

$$\lambda_t = \delta\lambda_{t+1}g'_t$$

where p_t is the market price, c'_t is the marginal harvest cost, and g'_t is the marginal future yield of stock in t. Thus, the market price of the harvested resource must cover both the marginal value of the unharvested resource and the marginal cost of harvesting it. Moreover, the value of one unit of resource today equals the discounted value of its yield tomorrow.

The steady-state resource stock s^*, harvest x^*, and shadow price λ^* solve the equation system

$$p(x^*) = c'(x^*) + \delta\lambda^* g'(s^* - x^*)$$

$$\lambda^* = \delta\lambda^* g'(s^* - x^*)$$

$$s^* = g(s^* - x^*).$$

These conditions imply $g'(s^* - x^*) - 1 = r$. That is, in steady-state, the marginal yield equals the interest rate.

Totally differentiating the equation system above:

$$\frac{\partial s^*}{\partial r} = \frac{1+r}{g''} < 0$$

$$\frac{\partial x^*}{\partial r} = -\frac{r}{g''} < 0$$

$$\frac{\partial \lambda^*}{\partial r} = \frac{(c'' - p')r}{g''} < 0.$$

That is, as the interest rate rises, the steady-state stock, the steady-state harvest, and the steady-state shadow price all fall.

Figure 8.4.5 Steady-state optimal harvest of a renewable resource.

8.4.6 Nonrenewable Resource Problem

A social planner wishes to maximize the discounted sum of net social surplus from harvesting a nonrenewable resource over an infinite horizon. For year t, let s_t denote the resource stock at the beginning of the year, let x_t denote the amount of the resource harvested, let $c_t = c(x_t)$ denote the total cost of harvesting, and let $p_t = p(x_t)$ denote the market clearing price. What is the socially optimal harvest policy?

This is an infinite horizon, deterministic model with time $t \in \{0, 1, 2, \ldots\}$ measured in years. There is one state variable,

$$\begin{aligned} s_t &= \text{stock of resource at beginning of year } t \\ s_t &\in [0, \infty), \end{aligned}$$

and one action variable,

$$x_t = \text{amount of resource harvested in year } t,$$

subject to the constraint

$$0 \leq x_t \leq s_t.$$

The reward earned by the optimizing agent is

$$\int_0^{x_t} p(\xi)\, d\xi - c(x_t).$$

State transitions are governed by

$$s_{t+1} = s_t - x_t.$$

The value function, which gives the net social value of resource stock, satisfies Bellman's equation

$$V(s) = \max_{0 \leq x \leq s} \{ \int_0^x p(\xi)\, d\xi - c(x) + \delta V(s - x) \}, \qquad s \geq 0.$$

Assuming $p(0) = \infty$ and $g(0) = 0$, the solution to the optimization problem embedded in Bellman's equation will be internal. Under these assumptions, the shadow price function satisfies the Euler conditions, which stipulate that for every stock level $s > 0$ there is a harvest level x such that

$$p(x) = c'(x) + \delta\lambda(s - x)$$

Figure 8.4.6 Optimal harvest path of a norenewable resource.

$$\lambda(s) = \delta\lambda(s - x).$$

Thus, along the optimal path

$$p_t = c_t' + \lambda_t$$

$$\lambda_t = \delta\lambda_{t+1}$$

where p_t is the market price and c_t' is the marginal harvest cost at t. That is, the market price of the harvested resource equals the marginal value of the unharvested resource plus the marginal cost of harvesting it. Also, the price of the harvested resource grows at the rate of interest. The steady-state, which occurs when stock is $s^* = 0$, is an uninteresting case.

8.4.7 Feedstock Problem

An animal weighing s_1 pounds in period $t = 1$ is to be fed up to period $T+1$, at which time it will be sold at a price of p dollars per pound. The cost of increasing the animal's weight by an amount x_t during period t is given by $c(s_t, x_t)$ where s_t is the animal's weight at the beginning of t. What feeding strategy maximizes the present value of profit?

This is a finite horizon, deterministic model with time $t \in \{1, 2, \ldots, T\}$ measured in feeding periods. There is one state variable,

$$\begin{aligned} s_t &= \text{weight of animal at beginning of period } t \\ s_t &\in [0, \infty), \end{aligned}$$

and one action variable,

$$x_t = \text{weight gain during period } t,$$

subject only to a nonnegativity constraint.

The reward earned by the hog farmer in feeding periods is

$$-c(s_t, x_t).$$

State transitions are governed by

$$s_{t+1} = s_t + x_t.$$

Figure 8.4.7 Feedstock problem dynamics.

The value function, which gives the value of animal weighing pounds s in period t, satisfies Bellman's equation

$$V_t(s) = \max_{x \geq 0}\{-c(s,x) + \delta V_{t+1}(s+x)\},$$

subject to the terminal condition

$$V_{T+1}(s) \equiv ps.$$

The shadow price function, which measures the price of animal mass, satisfies the Euler conditions, which stipulate that for each decision period t and weight level $s > 0$, the optimal weight gain x satisfies the complementarity conditions

$$x \geq 0$$

$$\delta\lambda_{t+1}(s+x) - c_x(s,x) le 0$$

$$x > 0 \Longrightarrow \delta\lambda_{t+1}(s+x) - c_x(s,x) = 0$$

$$\lambda_t(s) = -c_s(s,x) + \delta\lambda_{t+1}(s+x).$$

For the post-terminal period,

$$\lambda_{T+1}(s) = p.$$

Thus, along an optimal path, assuming an internal solution, we have:

$$\delta\lambda_{t+1} = c_x(s_t, x_t)$$

$$c_s(s_t, x_t) = \lambda_t - \delta\lambda_{t+1}.$$

In other words, the marginal cost of feeding the animal this period must equal the discounted value of the additional body mass obtained the following period. Also, the marginal value of body mass declines at the same rate at which it weight gains become increasingly more costly.

8.4.8 A Production-Adjustment Problem

The output price faced by a competitive firm follows a first-order autoregressive process:

$$p_{t+1} = \alpha + \gamma p_t + \epsilon_{t+1}, \qquad |\gamma| < 1, \epsilon_t \text{ i.i.d.}$$

The cost of producing q_t units in period t is $c(q_t)$ plus an adjustment cost of $a(q_t - q_{t-1})$. The firm cannot store the commodity because it is perishable. Assuming p_t is known at the time the period t production decision is made, what production policy maximizes the sum of current and expected future profits?

This is an infinite horizon, stochastic model with time $t \in \{0, 1, 2, \ldots\}$ measured in years. There are two state variables:

q_{t-1} = past production,

p_t = current market price.

There are one action variable:

q_t = current production

subject to a nonnegativity constraint.

The reward earned by the optimizing agent is

$$p_t \cdot q_t - c(q_t) - a(q_t - q_{t-1}).$$

Price state transitions are governed by

$$p_{t+1} = \alpha + \gamma p_t + \epsilon_{t+1}$$

where

ϵ_t = price process innovation in year t.

The transition rule for q_t is trivial.

The value function, which gives the value of the firm, satisfies Bellman's equation

$$V(q_{-1}, p) = \max_{0 \le q} \{pq - c(q) - a(q - q_{-1}) + \delta EV(q, \alpha + \gamma p + \epsilon)\}.$$

Assuming a positive production level, the Euler conditions require that for every state vector (q_{-1}, p), there is a production level q such that

$$p - c'(q) - a'(q - q_{-1}) + \delta E V_q(q, \alpha + \gamma p + \epsilon) = 0$$

$$V_q(q_{-1}, p) = a'(q - q_{-1})$$

Along the optimal path,

$$p_t = c'_t + (a'_t - \delta E a'_{t+1}).$$

Thus, marginal revenue equals the marginal production cost plus the net marginal adjustment cost.

The certainty-equivalent deterministic problem is obtained by assuming p is fixed at its longrun mean $\alpha/(1-\gamma)$. If $a'(0) = 0$, then the certainty-equivalent steady-state production is constant and implicitly defined by the short-run profit maximization condition:

$$p = c'(q)$$

8.4.9 A Production-Inventory Problem

The output price faced by a competitive firm follows a first-order autoregressive process:

$$p_{t+1} = \alpha + \gamma p_t + \epsilon_{t+1}, \qquad |\gamma| < 1, \epsilon_t \text{ i.i.d.}$$

The cost of producing q_t units in period t is $c(q_t)$. The firm may store across periods at a constant unit cost k. Assuming p_t is known at the time the period t production-inventory decision is made, what production-inventory policy maximizes the sum of current and expected future profits?

This is an infinite horizon, stochastic model with time $t \in \{0, 1, 2, \ldots\}$ measured in years. There are two state variables:

b_t = beginning inventories,

p_t = current market price.

There are two action variables:

q_t = current production

x_t = ending inventories

subject to the constraints

$q_t \geq 0$

$x_t \geq 0$

$x_t \leq q_t + b_t$;

that is, production, inventories, and deliveries must be nonnegative.

The reward earned by the optimizing agent is

$$p_t \cdot (q_t + b_t - x_t) - c(q_t) - kx_t.$$

State transitions are governed by

$$p_{t+1} = \alpha + \gamma p_t + \epsilon_{t+1}$$

where

ϵ_t = price process innovation in year t

and

$$b_{t+1} = x_t.$$

The value function, which gives the value of firm given inventories b and price p satisfies Bellman's equation

$$V(b,p) = \max_{0 \leq q, 0 \leq x \leq q+b} \{p(q+b-x) - c(q) - kx + \delta EV(x, \alpha + \gamma p + \epsilon)\}.$$

The shadow price function

$\lambda(b,p) = V_b(b,p)$ = marginal value of inventories

satisfies the Euler conditions, which require that for every beginning inventory level b and price p, there is a production level q, ending inventory level x, and material balance shadow price μ such that

$x \geq 0$

$\delta E\lambda(b, \alpha + \gamma p + \epsilon) - p - k - \mu \leq 0$

$$x > 0 \Longrightarrow \delta E\lambda(b, \alpha + \gamma p + \epsilon) - p - k - \mu = 0$$

$$q \geq 0$$

$$p - c'(q) \leq 0$$

$$q > 0 \Longrightarrow p - c'(q) = 0$$

$$\mu \geq 0$$

$$q + b - x \leq 0$$

$$\mu > 0 \Longrightarrow q + b - x = 0$$

$$\lambda(b, p) = p - \mu$$

Along the optimal path, if deliveries and storage are positive,

$$\delta E_t p_{t+1} - p_t - k = 0$$

$$p_t = c'_t.$$

That is, marginal revenue equals the marginal production cost and the discounted expected future price equals the current output price plus the cost of storage.

The certainty-equivalent deterministic problem is obtained by assuming p is fixed at its longrun mean $\alpha/(1-\gamma)$. The certainty-equivalent steady-state inventories are 0 and production is constant and implicitly defined by the short-run profit maximization condition:

$$p = c'(q).$$

8.4.10 Optimal Growth with Debt

Reconsider the optimal growth problem when the central planner can carry an external debt load d_t whose unit cost $\eta_0 + \eta_1 q_t$ rises with the debt to asset ratio $q_t = d_t/s_t$.

This is an infinite horizon, stochastic model with time $t \in \{0, 1, 2, \ldots\}$ measured in years. There are two state variables:

$$\begin{aligned} s_t &= \text{stock of good at beginning of year } t \\ s_t &\in [0, \infty) \end{aligned}$$

and

$$\begin{aligned} d_t &= \text{debt load at beginning of year } t \\ d_t &\in (-\infty, \infty). \end{aligned}$$

Here, $d_t < 0$ implies that the economy runs a surplus. There are two action variables:

$$x_t = \text{amount of good invested in year } t$$

$$c_t = \text{amount of good consumed in year } t,$$

both subject to nonnegativity constraints.

The reward earned by the optimizing agent is

$$u(c) = \text{social utility in } t.$$

Supply state transitions are governed by

$$s_{t+1} = \gamma x_t + \epsilon_{t+1} f(x_t)$$

where

$$\epsilon_t = \text{productivity shock in year } t.$$

Debt state transitions are governed by

$$d_{t+1} = d_t + b_t,$$

where

$$b_t = c_t + x_t + (\eta_0 + \eta_1 d_t / s_t) \cdot d_t - s_t,$$

indicates net borrowing in year t.

The value function, which gives the sum of current and expected future social welfare, satisfies Bellman's equation

$$V(s,d) = \max_{x \geq 0, c \geq 0} \{u(c) + \delta EV(\gamma x + \epsilon f(x), d + b)\}$$

where $b = x + c + (\eta_0 + \eta_1 d/s) \cdot d - s)$ is net borrowing.

Assuming $u'(0) = -\infty$ and $f(0) = 0$, the solution to Bellman's equation will always be internal, and the shadow price and cost functions

$$\lambda(s,d) = \frac{\partial V}{\partial s}(s,d) = \text{shadow price of stock}$$

and

$$\mu(s,d) = \frac{\partial V}{\partial d}(s,d) = \text{shadow cost of debt}$$

satisfy the Euler equilibrium conditions, which stipulate that for every stock level $s > 0$ and debt level d,

$$u'(c) + \delta E\mu(\gamma x + \epsilon f(x), d + b) = 0$$

$$\delta E\left[\lambda(\gamma x + \epsilon f(x), d + b) \cdot (\gamma + \epsilon f'(x))\right] + \delta E\mu(\gamma x + \epsilon f(x), d + b) = 0$$

$$\lambda(s,d) = -\delta E[\mu(\gamma x + \epsilon f(x), d + b) \cdot (1 + \eta_1 q^2)]$$

$$\mu(s,d) = \delta E[\mu(\gamma x + \epsilon f(x), d + b) \cdot (1 + \eta_0 + 2\eta_1 q)]$$

where $q = d/s$ is the debt to asset ratio.

The certainty-equivalent steady-state is obtained by assuming $\epsilon = 1$ with probability 1. The certainty-equivalent steady-state stock of good s^*, debt load d^*, debt-asset ratio $q^* = d^*/s^*$, investment level x^*, consumption level c^*, stock shadow price λ^*, and debt shadow cost μ^* solve the equation system

$$u'(c) + \delta\mu^* = 0$$

$$\delta\lambda^*(\gamma + f'(x^*)) + \delta\mu^* = 0$$

$$\lambda^* = -\delta\mu^*(1 + \eta_1 q^{*2})$$

$$\mu^* = \delta\mu^*(1 + \eta_0 + 2\eta_1 q^*)$$

$$s^* = \gamma x^* + f(x^*)$$

$$s^* = x^* + c^* + (\eta_0 + \eta_1 q^*)d^*$$

$$q^* = d^*/s^*.$$

These conditions imply a steady-state optimal debt load $q^* = (r - \eta_0)/(2\eta_1)$, which increases with the discount rate r but falls with the base cost of debt η_0.

8.5 Rational Expectations Models

By definition, agents in rational expectations models take into account how their actions will affect them in the future and form expectations that coincide with those implied by the model as a whole. Most discrete time rational expectation models take the following form: At the beginning of period t, an economic system emerges in a state s_t. The agents in the economic system observe the state of the system and, by pursuing their individual objectives, formulate a collective behavioral response x_t. The economic system then evolves to a new state s_{t+1} that depends on the current state s_t and response x_t, and an exogenous random shock ϵ_{t+1} that is realized only after the agents respond at time t.

More formally, the behavioral responses of economic agents and the state transitions of the economic system are governed by a structural law of the form

$$f(s_t, x_t, E_t x_{t+1}) = 0,$$

and the dynamic law

$$s_{t+1} = g(s_t, x_t, \epsilon_{t+1}).$$

The stipulation that only the expectation of the subsequent period's behavioral response is relevant to the current response of agents is more general than first appears. By introducing new accounting variables, the current response can be made to depend on the expectation of any function of future states and responses, including states and responses more than one period into the future.

The state space $S \in \Re^n$, which contains all the states attainable by the economic system, and the response space $X \in \Re^m$, which contains all behavioral responses that may be made by the economic agents, are both assumed to be closed convex nonempty sets. In some instances, the range of admissible responses may vary with the state of the process s_t. In such cases, the restricted response space will be denoted $X(s_t)$ and will be assumed to be a closed convex nonempty set. The structure f and dynamic law g are assumed to be twice continuously differentiable on S and X and the per-period discount factor δ is assumed to be less than one. The exogenous random shocks ϵ_t are assumed identically distributed over time, mutually independent, and independent of past states and responses.

The primary task facing an economic analyst is to explain the behavioral response $x = x(s)$ of agents in each state s attainable by the process. The response function $x(\cdot)$ is characterized implicitly as the solution to a functional equation:

$$f(s, x(s), Ex(g(s, x(s), \epsilon)) = 0 \qquad \forall s \in S.$$

In many instances, this functional equation will not possess a closed-form solution and can only be solved numerically.

8.5.1 Lucas-Prescott Asset Pricing Model

The basic rational expectations asset pricing model has been studied extensively by macroeconomists. The model assumes the existence of a pure exchange economy in which a representative infinitely-lived agent allocates real wealth between immediate consumption q_t and investment in an index asset i_t. The agent's objective is to maximize expected lifetime utility subject to an intertemporal budget constraint:

$$\begin{aligned} \max \quad & E_t\{\sum_{k=0}^{\infty} \delta^k u(q_{t+k})\} \\ \text{s.t.} \quad & q_t + i_t = i_{t-1} r_t. \end{aligned} \tag{8.1}$$

Here, E_t is the conditional expectation operator given information available at time t, δ is the agent's subjective discount rate, i_t is the amount of asset held by the agent at the end of period t, and r_t is the asset's return in period t.

Under mild regularity conditions, the agent's dynamic optimization problem has an unique solution that satisfies the first-order Euler condition:

$$\delta E_t[u'(q_{t+1}) r_{t+1}] = u'(q_t).$$

The Euler condition asserts that along an optimal consumption path the marginal utility of consuming one unit of wealth today equals the marginal benefit of investing the unit of wealth and consuming it and its dividends tomorrow.

The asset pricing model may be completed by specifying the utility function, introducing a production sector, and imposing a market clearing con-

dition. Assume that the agent's preferences exhibit constant relative risk-aversion $\gamma > 0$:

$$u(q) = \frac{q^{1-\gamma}}{1-\gamma}.$$

We assume that aggregate output y_t is exogenous and follows a stationary first-order autoregressive process whose innovation ϵ_t is normally distributed white noise with standard deviation σ_ϵ:

$$y_t = \alpha + \beta y_{t-1} + \epsilon_t.$$

And we assume that output is consumed entirely in the period that it is produced:

$$y_t = q_t.$$

A formal solution to the rational expectations asset pricing model is a rule that gives the equilibrium asset return r_t as a function of current and past realizations of the driving exogenous output process. Lucas demonstrated that when the output process is stationary and first-order Markovian, as assumed here, the rule is well-defined. In particular, the equilibrium return in period t will be a stationary deterministic function of the contemporaneous output level y_t:

$$r_t = \lambda(y_t).$$

From the dynamic equilibrium conditions, it follows that the asset return function λ is characterized by the equilibrium condition:

$$E_\epsilon \delta(\alpha + \beta y + \epsilon)^{-\gamma} \lambda(\alpha + \beta y + \epsilon) = y^{-\gamma} \qquad \forall y.$$

The Euler functional equation of the asset pricing model is nonlinear and lacks a known a closed-form solution. It can only be solved approximately using numerical functional equation methods.

8.5.2 Competitive Storage Under Uncertainty

The centerpiece of the classical theory of storage is the competitive intertemporal arbitrage equation

$$\delta E_t p_{t+1} - p_t = c(x_t).$$

The intertemporal arbitrage equation asserts that, in equilibrium, expected appreciation in the commodity price p_t must equal the unit cost of storage $c(x_t)$. Dynamic equilibrium in the commodity market is enforced by competitive expected-profit-maximizing storers. Whenever expected appreciation exceeds the storage cost, the attendant profits induce storers to increase their stockholdings until the equilibrium is restored. Conversely, whenever the storage cost exceeds expected appreciation, the attendant loses induce storers to decrease their stockholdings until the equilibrium is restored.

According to the classical theory, the unit storage cost $c(x_t)$ is a nondecreasing function of the amount stored x_t. The unit storage cost represents the marginal physical cost of storage less the marginal "convenience yield", which is the amount processors are willing to pay to have sufficient stocks available to avoid costly production adjustments. If stock levels are high, the marginal convenience yield is zero and the unit storage cost equals the physical storage cost. As stock levels approach zero, however, the marginal convenience yield rises, eventually resulting in a negative unit storage cost. The classical storage model has received strong empirical support over the years and captures the key stylized fact of markets for storable commodities: the coincidence of negative intertemporal price spreads and low, but positive, stock levels.

The modern theory of storage extends the classical model to a partial equilibrium model of price-quantity determination by appending supply, demand, and market clearing conditions to the intertemporal arbitrage equation. For the sake of discussion, let us consider a simple agricultural commodity market model with exogenous production. Denote quantity consumed by q_t, quantity harvested h_t, available supply by s_t, and the per-period discount factor by δ. Assume that the market clearing price is a decreasing function of the quantity consumed:

$$p_t = p(q_t);$$

that available supply is either consumed in the current period or stored:

$$s_t = q_t + x_t;$$

and that the supply available next period will be the sum of current carryout and next period's harvest:

$$s_{t+1} = x_t + h_{t+1}.$$

Figure 8.5.2 Solution to Rational Expectations Storage Model

The modern storage model is closed by assuming that price expectations are consistent with the other structural assumptions of the model. The so-called rational expectations assumption endogenizes the expected future price while preserving internal consistency of the model.

The solution of the nonlinear rational expectations commodity market model is illustrated in Figure 8.5.2. These figures show, respectively, equilibrium price and carryout in terms of available supply. For comparison, the first figure also shows the inverse consumption demand function $p(\cdot)$, which gives the market price that would prevail in the absence of storage. At low supply levels, there is effectively no storage and the equilibrium price coincides with the inverse consumption demand function. Over this range, acreage supply is not significantly affected by variations in available supply. At sufficiently high supply levels, incentives for speculative storage begin to appear. Over this range, the equilibrium price, which reflects both consumption and storage demand, exceeds the inverse consumption demand function.

The nonlinear rational expectations commodity market model cannot be solved using standard algebraic techniques. To see this, let $\lambda(s)$ denote the equilibrium price implied by the model for a given available supply s. Having the equilibrium price function $\lambda(\cdot)$, the rational ex-ante expected price could be computed by integrating over the harvest distribution:

$$E_t p_{t+1} = E_y \lambda(x_t + h_{t+1})$$

Appending this equation to the previous three market equations would result in a system of four nonlinear algebraic equations that in principle could be solved for all the unknowns.

Unfortunately, the equilibrium price function $\lambda(\cdot)$ is not known a priori and deriving it, the key to solving the commodity market model, is a nontrivial functional equation problem. Combining all the behavioral relations, we see that $\lambda(\cdot)$ must simultaneously satisfy an infinite number of conditions. Specifically, for every realizable supply s,

$$\lambda(s) = p(s - x)$$

where stock x solves

$$\delta E_y \lambda(x + h) - p(s - x) = c(x)$$

In the general framework developed for rational expectations models above, available supply is only state variable, price and carryout are the response variables, and harvest is the random driving shock. Only the relationship between price and supply needs to be derived, since only future price expectations affect behavior, and once the price and supply are known, the carryout may be computed from the inverse demand function.

An alternative way to pose the rational expectations commodity storage model is to integrate it into an equivalent optimization problem. Consider the problem of maximizing the discounted expected sum of consumer surplus less storage costs. The resulting dynamic optimization problem, with state variable s and action variable x, yields the following Bellman equation:

$$V(s) = \max_{0 \le x \le s} \{ \int_0^{s-x} p(\xi)\, d\xi - \int_0^x c(\xi)\, d\xi + \delta EV(x+h) \}, \qquad s \ge 0.$$

One may verify that the Euler equilibrium conditions for this dynamic optimization problem are precisely the equilibrium conditions of the original rational expectations model, provided that the shadow price of the optimization problem is identified with the rational expectations equilibrium market price.

Finally, one might compute certainty-equivalent steady-state supply s^*, storage x^*, and price p^* by solving the equation system

$$\delta p^* = f(s^* - x^*) + c(x^*)$$

$$p^* = f(s^* - x^*)$$

$$s^* = x^* + h^*$$

where h^* is the expected harvest.

8.6 Dynamic Games

Dynamic game models attempt to capture strategic interactions among a small number of dynamically optimizing agents when the actions of one agent affects the welfare of another. For the sake of brevity, we consider only two agent games. The theory and methods developed below, however, can be generalized to accommodate an arbitrary number of agents.

Denote by s_i the state of the process controlled by agent i, and denote by x_i the action taken by agent i. In a dynamic game setting, agent i receives a reward that depends not only on the state of his own process and the action he takes, but also the state s_j of the other agent's process and the action x_j that he takes. Specifically, the reward earned by agent i at any point in time is $f_i(s_i, s_j, x_i, x_j)$.

As with a static game, the equilibrium solution to a dynamic game depends on the information available to the agents and the class of strategies they are allowed to pursue. For simplicity, we consider only the most common game structure. Specifically, we will seek a noncooperative Nash game equilibrium under the assumption that each agent knows the other agent's state at any point in time, and that each agent also knows the policy followed by the other agent. A dynamic Nash game equilibrium exists when each agent's policy maximizes his own stream of current and expected future rewards given that the other agent follows his policy.

The dynamic Nash game equilibrium may be formally expressed by a pair of Bellman equations, one for each agent. The Bellman equation for agent i takes the form

$$V_i(s_i, s_j) = \max_{x \in X(s_i, s_j)} \{f_i(s_i, s_j, x_i, x_j) + \delta E_\epsilon V_i(g_1(s_1, x_1, \epsilon_1), g_2(s_2, x_2, \epsilon_2))\},$$

for $s_i, s_j \in S$. Here, $V_i(s_i, s_j)$ denotes the maximum current and expected future rewards that can be earned by agent i, given that agent j remains committed to his policy. Solving for the Nash equilibrium involves finding policies x_i and x_j for every state that solve the Bellman equations of both agents simultaneously.

Let λ_{ii} denote the partial derivative of agent i's value function with respect to the state controlled by him:

$$\lambda_{ii}(s_1, s_2) = \frac{\partial V_i}{\partial s_i}(s_1, s_2) \qquad \forall s_1, s_2.$$

Also, let λ_{ij} denote the partial derivative of agent i's value function with respect to the state controlled by agent j:

$$\lambda_{ij}(s_1, s_2) = \frac{\partial V_i}{\partial s_j}(s_1, s_2) \qquad \forall s_1, s_2.$$

The shadow price function λ_{ii} represents agent i's valuation of a marginal unit of the state controlled by him; the shadow price function λ_{ij} represents agent i's valuation of a marginal unit of the state controlled by his rival.

The first-order equilibrium conditions for the Nash dynamic game are derived by applying the Karush-Kuhn-Tucker and Envelope Theorems to the optimization problems embedded in the two Bellman equations. Assuming actions are unconstrained, the Karush-Kuhn-Tucker conditions for the embedded unconstrained optimization problems imply that the optimal action x_i for agent i, given state s_i, s_j, must satisfy the equimarginality condition:

$$\frac{\partial f_i}{\partial x_i}(s_1, s_2, x_1, x_2) + \delta E_\epsilon \left[\lambda_{ii}(s_1', s_2') \frac{\partial g_i}{\partial x_i}(s_i, x_i, \epsilon_i) \right] = 0$$

where $s_i' = g_i(s_i, x_i, \epsilon_i)$. The Envelope Theorem applied to the same problem implies:

$$\lambda_{ii}(s_1, s_2) = \frac{\partial f_i}{\partial s_i}(s_1, s_2, x_1, x_2) + \frac{\partial f_i}{\partial x_j}(s_1, s_2, x_1, x_2) \frac{\partial x_j}{\partial s_i}(s_1, s_2) +$$

$$\delta E_\epsilon \left[\lambda_{ii}(s_1', s_2') \frac{\partial g_i}{\partial s_i}(s_i, x_i, \epsilon_i) + \lambda_{ij}(s_1', s_2') \frac{\partial g_j}{\partial x_j}(s_j, x_j, \epsilon_j) \frac{\partial x_j}{\partial s_i}(s_1, s_2) \right]$$

$$\lambda_{ij}(s_1, s_2) = \frac{\partial f_i}{\partial s_j}(s_1, s_2, x_1, x_2) + \frac{\partial f_i}{\partial x_j}(s_1, s_2, x_1, x_2) \frac{\partial x_j}{\partial s_j}(s_1, s_2) +$$

$$\delta E_\epsilon \left[\lambda_{ij}(s_1', s_2') \frac{\partial g_j}{\partial s_j}(s_j, x_j, \epsilon_j) + \lambda_{ij}(s_1', s_2') \frac{\partial g_j}{\partial x_j}(s_j, x_j, \epsilon_j) \frac{\partial x_j}{\partial s_j}(s_1, s_2) \right].$$

The Euler conditions for a two agent dynamic game thus comprise six functional equations in six unknown functions: the two own-shadow price functions, the two cross-shadow price functions, and the two optimal policy functions.

8.6.1 Risk Sharing Game

Consider an economy comprising two agents and a single composite good. Each year t begins with predetermined amounts of the good s_{1t} and s_{2t} held by the two agents, respectively. Given the amounts on hand, each agent selects an amount x_{it} to be invested, and consumes the rest. The utility derived from consumption in year t by agent i is $u_i(s_{it} - x_{it})$. Given each agent's investment decision, the amount of good available in year $t+1$ to agent i will be $s_{i,t+1} = g_i(x_{it}, \epsilon_{i,t+1}) = \gamma x_{it} + \epsilon_{i,t+1} f_i(x_{it})$ where γ is the

capital survival rate, f_i is agent i's production function, and $\epsilon_{i,t+1}$ is a positive production shock with mean 1 that is specific to agent i.

Suppose now that the two agents agree to insure against a string of production disasters by entering into a contract to share collective wealth in perpetuity. Specifically, the agents agree that, in any given period t, the wealthier of the two agents will transfer a certain proportion σ of the wealth differential to the poorer agent. Under this scheme, if agent i begins period t with wealth s_{it}, his post-transfer wealth will be $\hat{s}_{it} = s_{it} - \sigma(s_{it} - s_{jt})$. If the wealth transfer is enforceable, but agents are free to pursue consumption and investments freely, moral hazard will arise. In particular, both agents will have incentives to change their consumption and investment policies upon introduction of insurance. How will insurance affect the agents' investment behavior, and for what initial wealth states s_{1t} and s_{2t} and share parameter σ will both agents be willing to enter into the insurance contract?

The essence of the dynamic Nash game equilibrium for the redistribution game is captured by a pair of Bellman equations, one for each agent. The Bellman equation for agent i takes the form

$$V_i(s_i, s_j) = \max_{0 \leq x_i \leq \hat{s}_i} \{u_i(\hat{s}_i - x_i) + \delta E_\epsilon V_i(g_i(x_i, \epsilon_i), g_j(x_j, \epsilon_j))\},$$

where $\hat{s}_i = s_i - \sigma(s_i + s_j)$, for $s_i, s_j \in S$. Here, $V_i(s_i, s_j)$ denotes the maximum current and expected future rewards that can be earned by agent i, given that agent j remains committed to his policy.

The first-order equilibrium conditions for the Nash dynamic game are derived by applying the Karush-Kuhn-Tucker and Envelope Theorems to the optimization problems embedded in the two Bellman equations. Assuming an internal solution to each agent's investment problem, the Karush-Kuhn-Tucker conditions imply that the optimal investment x_i for agent i, given wealths s_i, s_j, must satisfy the equimarginality condition:

$$-u_i'(\hat{s}_i - x_i) + \delta E_\epsilon \left[\lambda_{ii}(s_1', s_2') \frac{\partial g_i}{\partial x_i}(x_i, \epsilon_i)\right] = 0$$

where $s_i' = g_i(x_i, \epsilon_i)$. The Envelope Theorem applied to the same problem implies:

$$\lambda_{ii}(s_1, s_2) = (1 - \sigma)u_i'(\hat{s}_i - x_i) + \delta E_\epsilon \left[\lambda_{ij}(s_1', s_2') \frac{\partial g_j}{\partial x_j}(x_j, \epsilon_j) \frac{\partial x_j}{\partial s_i}(s_1, s_2)\right]$$

$$\lambda_{ij}(s_1, s_2) = \sigma u_i'(\hat{s}_i - x_i) + \delta E_\epsilon \left[\lambda_{ij}(s_1', s_2') \frac{\partial g_j}{\partial x_j}(x_j, \epsilon_j) \frac{\partial x_j}{\partial s_j}(s_1, s_2)\right],$$

The Euler conditions for a two agent dynamic game thus comprise six functional equations in six unknown functions: the two own-shadow price functions, the two cross-shadow price functions, and the two optimal policy functions.

8.6.2 Marketing Board Game

Assume that there are only two countries that can supply a given commodity on the world market. In each country, a government marketing board has the exclusive power to sell the commodity on the world market. The marketing boards compete with each other, using storage as a strategy variable to maximize the present value of current and expected future income from commodity sales.

For each exporting country $i = 1, 2$ and period t, let s_{it} denote the supply available at the beginning of period, let q_{it} denote the quantity exported, let x_{it} denote the stocks held at the end of the period, let y_{it} denote new production, let p_t denote the world price, let c_{it} denote total storage costs, and let δ denote the discount factor.

Formally, each marketing board $i = 1, 2$, solves

$$\max E \sum_{t=0}^{\infty} \delta[p_t q_{it} - c_{it}]$$

subject to the following conditions: Available supply is the sum of beginning stocks and new production:

$$s_{it} = x_{it-1} + y_{it}.$$

Available supply is either exported or stored:

$$s_{it} = q_{it} + x_{it}.$$

The world market clearing price p_t is a decreasing function $\pi(\cdot)$ of the total amount exported:

$$p_t = \pi(q_{1t} + q_{2t}).$$

The cost of storage is an increasing function $c_i(\cdot)$ of the quantity stored:

$$c_{it} = c_i(x_{it}).$$

And production y_{it} is exogenous, stochastic, independently distributed across countries, and independently and identically distributed across time.

Each marketing board faces a dynamic optimization problem subject to the constraints. The price, and thus the payoff, for each country at time t is simultaneously determined by the quantities marketed by both boards. In making its storage decision, each board must anticipate the storage decision of its rival. The two optimization problems must therefore be solved simultaneously to determine the equilibrium levels of stocks, exports, and price.

The noncooperative Nash equilibrium is characterized by a pair of Bellman equations, which for country i takes the form

$$V_i(s_1, s_2) = \max_{x_i}[pq_i - c_i + \delta E_y V_i(x_1 + y_1, x_2 + y_2)] \qquad \forall s_1, s_2$$

where $q_i = s_i - x_i$, $p = \pi(q_1 + q_2)$, and $c_i = c_i(x_i)$.

For each combination of $i = 1, 2$ and $j = 1, 2$, let λ_{ij}^* denote the partial derivative of country i's value function with respect to the supply in country j:

$$\lambda_{ij}(s_1, s_2) = \frac{\partial V_i}{\partial s_j}(x_1 + y_1, x_2 + y_2) \qquad \forall s_1, s_2.$$

The shadow price function λ_{ij} represents country i's valuation of a marginal unit of stock in country j.

Applying the to Envelope Theorem to Bellman equation, the own shadow price function must satisfy

$$\lambda_{ii}(s_1, s_2) = p + p' q_i[1 - \frac{\partial x_j}{\partial s_i}] + \delta E_y \lambda_{ij}(x_1 + y_1, x_2 + y_2)\frac{\partial x_j}{\partial s_i} \qquad \forall s_1, s_2$$

and the cross shadow price function must satisfy

$$\lambda_{ij}(s_1, s_2) = p' q_i[1 - \frac{\partial x_j}{\partial s_j}] + \delta E_y \lambda_{ij}(x1 + y_1, x2 + y_2)\frac{\partial x_j}{\partial s_j} \qquad \forall s_1, s_2$$

where $p' = \pi'(q_1 + q_2)$.

Another necessary condition for the dynamic feedback Nash equilibrium can be obtained by deriving the first-order condition for the optimization problem embedded in Bellman's equation:

$$p + p' q_i = \delta E_y \lambda_{ii}(x_1 + y_1, x_2 + y_2) - c_i' \qquad \forall s_1, s_2$$

where $c_i' = c_i'(x_i)$. This condition asserts that along an equilibrium path, the marginal payoff from selling this period $p + p'q_i$ must equal the expected marginal payoff from storing and selling next period $\delta E_y \lambda_{ii} - c_i'$.

The noncooperative feedback Nash equilibrium for the game between the two marketing boards is characterized by six functional equations in six unknowns: the equilibrium feedback strategies x_1 and x_2 and the equilibrium shadow price functions λ_{11}, λ_{12}, λ_{21}, and λ_{22}.

Exercises

1. An industrial firm's profit in period t

   $$\pi(q_t) = \alpha_0 + \alpha_1 q_t - 0.5 q_t^2$$

   is a function of its output q_t. The firm's production process generates an environmental pollutant. Specifically, if x_t is the level of pollutant in the environment in period t, then the level of the pollutant the following period will be

   $$x_{t+1} = \beta x_t + q_t$$

   where $0 < \beta < 1$.

   During the Reagan-Bush administration, the firm operated without regard to environmental consequences at its profit maximizing level $q_t = \alpha_1$. You have been asked by a yet unindicted member of the Clinton administration to examine ways of inducing the firm to act in a more socially responsible manner. Net social welfare, according the Clinton administration, is given by

   $$\sum_{t=0}^{\infty} \delta^t \left[\pi(q_t) - c x_t\right]$$

   where c is the unit social cost of suffering the pollutant and $\delta < 1$ is the social discount factor.

   (a) Set up the social planner's decision problem of determining the stream of production levels that maximizes net social welfare. Specifically, formulate Bellman's equation, clearly identifying the states and actions, the reward function, the transition rule, and the value function.

(b) Assuming an internal solution, derive and interpret the Euler conditions for socially optimal production. What does the derivative of the value function represent?

(c) Solve for the steady-state socially optimal production level q^* and pollution level x^* in terms of the model parameters $(\alpha_0, \alpha_1, \delta, \beta, c)$.

(d) Determine the per-unit tax on output τ that will induce the firm to produce at the steady-state socially optimal production level q^*.

2. Consider the problem of harvesting a renewable resource over an infinite time horizon. For year t, let s_t denote the resource stock at the beginning of the year, let x_t denote the amount of the resource harvested, let $p_t = p(x_t) = \alpha_0 - \alpha_1 x_t$ denote the market clearing price, and let $c_t = c(s_t) = \beta_0 + \beta_1 s_t$ denote the unit cost of harvest. Assume an annual interest rate r and a stock growth dynamic $s_{t+1} = s_t + \gamma(\bar{s} - s_t) - x_t$ where $\bar{s}$ is the no-harvest steady-state stock level.

(a) Formulate and interpret the equilibrium conditions that characterize the optimal solution to the social planner's problem of maximizing the discounted sum of net social surplus over time.

(b) Formulate and interpret the equilibrium conditions that characterize the optimal solution to the monopolist's problem of maximizing the discounted sum of profits over time.

(c) In (a) and (b), explicitly solve the steady-state conditions for the steady-state harvest and stock levels, x^* and s^*. Does the monopolist or the social planner maintain the larger steady-state stock of resource?

(d) How do the steady-state equilibrium stock levels change if demand rises (i.e., if α_0 rises)? How do they change if the harvest cost rises (i.e., if β_0 rises)?

3. Consider the optimal management of a timber stand whose biomass at time t is S_t. The biomass transition function is described by

$$\ln S_{t+1}/S_t \sim N(\mu, \sigma^2).$$

The decision problem is to determine when to clear cut and replant the entire stand. The price obtained for cut timber is p dollars per unit

and the cost of replanting is c dollars. The period after cutting, the stand has biomass equal to one unit.

Please answer the following questions:

(a) Formulate and interpret Bellman's equation.

(b) What conditions characterize the certainty equivalent steady-state?

(c) How would you solve and simulate this model in order to gain an understanding of timber harvest dynamics.

4. Consider an aquaculturist that wishes to maximize the present value of profits derived from harvesting catfish grown in a pond. For period t, let s_t denote the quantity of catfish in the pond at the beginning of the period and let x_t denote the quantity of catfish harvested. Assume that the market price p of catfish is constant over time and that the total cost of harvesting in period t is given by $c_t = c(s_t, x_t) = \alpha x_t - \beta(s_t x_t - 0.5x_t^2)$. Assume an annual discount factor $\delta > 0$ and a stock growth dynamic $s_{t+1} = \gamma(s_t - x_t)$, where $\gamma > 1$.

   (a) Formulate and interpret the Bellman equation that characterizes the optimal harvest policy.

   (b) Formulate and interpret the Euler conditions that characterize the optimal harvest policy.

   (c) How does the steady-state stock level vary with the discount factor?

5. Consider a infinite-horizon, perfect foresight model

$$f(s_t, x_t, x_{t+1}) = 0$$

$$s_{t+1} = g(s_t, x_t)$$

where s_t and x_t denote, respectively, the state of the economy and the response of agents in the economy at time t.

   (a) How would you compute the steady-state (s^*, x^*) of the economic system?

   (b) How would you compute the function $x(\cdot)$, that relates the action of agents to the state of the economy: $x_t = x(s_t)$?

Describe your procedure in mathematical terms first, and then sketch out the key blocks of Matlab code that you would write to implement the procedure. You may use the functions that I wrote and distributed to you in class in your sketch.

6. At time t, a firm earns net revenue

$$\pi_t = py_t - rk_t - \tau_t k_t - c_t$$

where p is the market price, y_t is output, r is the capital rental rate, k_t is capital at the beginning of the period, c_t is the cost of adjusting capital, and τ_t is tax paid per unit of capital. The firm's production function, ajdustment costs, and tax rate are given by

$$\begin{aligned} y_t &= \alpha k_t, \\ c_t &= 0.5\beta(k_{t+1} - k_t)^2, \\ \tau_t &= \tau + 0.5\gamma k_t. \end{aligned}$$

Assume that the unit output price p and the unit capital rental rate r are both exogenously fixed and known; also assume that the parameters $\alpha > 0$, $\beta > 0$, $\gamma > 0$, and $\tau > 0$ are given. Formulate the firm's problem of maximizing the present value of net revenue over an infinite time horizon. Specifically:

   (a) Set up the decision problem (define states, actions, reward function, transition rule).

   (b) Formulate the value function and Bellman's recursive functional equation.

   (c) Assuming an internal solution, derive and interpret the first order conditions for optimality. What does the derivative of the value function represent?

   (d) What effect does an increase in the base tax rate, τ, have on output in the long run.

   (e) What effect does an increase in the discount factor, δ, have on output in the long run.

7. Consider the Optimal Growth example in this Chapter. Find and sign $\frac{\partial s^*}{\partial \gamma}$, $\frac{\partial x^*}{\partial \gamma}$, and $\frac{\partial \lambda^*}{\partial \gamma}$.

8. Consider the Optimal Growth with Debt example in this Chapter. Find the golden rule of growth with debt. Also, perform comparative statics analysis with respect to the steady state.

9. Consider the Renewable Resource example in this Chapter. However, now assume that the renewable resource is entirely owned by a profit-maximizing monopolist. Will the steady-state harvest and stock levels be greater for the monopolist or for the social planner? Give conditions under which a "regular" steady-state will exist. What if these conditions are not satisfied?

10. Hogs breed at a rate β. That is, if a farmer breeds x_t hogs during period t, there will be $(1+\beta)x_t$ hogs at the beginning of period $t+1$. At the beginning of any period, hogs can be marketed for a profit p per hog. Only the hogs not sent to market at the beginning of the period are available for breeding during the period. A farmer has H hogs at the beginning of period 0. Find the hog marketing strategy that maximizes the present value of profits over a T-period horizon.

11. A firm has a contractual obligation to deliver Q units of its product to a buyer firm at the beginning of period T; that is, letting x_t denote inventories on hand at the beginning of period t, the firm must produce sufficient quantities in periods $0, 1, 2, \ldots, T-1$ so as to ensure that $x_T \geq Q$. The cost of producing q_t units in period t is given by $c(q_t)$, where $c' > 0$. The unit cost of storage is k dollars per period; due to spoilage, a proportion β of inventories held at the beginning of one period do not survive to the following period. The firm's initial inventories are x_0 where $0 < x_0 < Q$. The firm wishes to minimize the present value of the cost of meeting its contractual obligation; assume a discount factor $\delta < 1$.

    (a) Identify the state and decision variables, the payoff function, and the equation of motion associated with this problem.

    (b) Write Bellman's recursive equation. What does the value function represent?

    (c) Derive the first order conditions for optimality and interpret them. What does the derivative of value function represent?

    (d) Assuming increasing marginal cost, $c'' > 0$, qualitatively describe the optimal production plan.

(e) Assuming decreasing marginal cost, $c'' < 0$, qualitatively describe the optimal production plan.

12. A subsistence farmer grows and eats a single crop. Production, y_t, depends on how much seed is on hand at the beginning of the year, k_t, according to

$$y_t = k_t^\alpha$$

where $0 < \alpha < 1$.

The amount kept for next year's seed is the difference between the amount produced and the amount consumed, c_t:

$$k_{t+1} = y_t - c_t.$$

The farmer has a time-additive logarithmic utility function and seeks to maximize

$$\sum_{t=0}^{T} \delta^t \ln(c_t).$$

subject to having an initial stock of seed, k_0. What is the farmer's optimal consumption-investment policy?

(a) Set up the decision problem (define states, decisions, objective function, transition equation).

(b) Formulate the value function and Bellman's recursive functional equation.

(c) Derive and interpret the first order conditions for optimality.

(d) Show that the value function is time invariant and has the form

$$V(k_t) = A + B \ln(k_t)$$

and that the optimal decision rule for this problem is

$$k_{t+1} = C y_t;$$

find the values for A, B, and C.

13. A firm competes in a mature industry whose total profit is a fixed amount X every year. If the firm captures a fraction p_t of total industry sales in year t, it makes a profit $p_t X$. The fraction of sales captured by the firm in year t is a function $p_t = f(p_{t-1}, a_{t-1})$ of the fraction it captured the preceding year and its advertising expenditures the preceding year, a_{t-1}. Find the advertising policy that maximizes the firm's discounted profits over a fixed time horizon of T years. Assume p_0 and a_0 are known.

    (a) Set up the decision problem (define states, decisions, objective function, transition equation).

    (b) Formulate the value function and Bellman's recursive functional equation.

    (c) Derive and interpret the first order conditions for optimality.

    (d) Assuming an infinite horizon, what conditions characterize the steady-state optimal solution?

14. A corn producer's net per-acre revenue in year t is given by

$$c_t = p_t y_t - c_t x_t - w_t l_t$$

    where p_t is the unit price of corn (\$/bu.), y_t is the corn yield (bu./acre), c_t is the unit cost of fertilizer (\$/lb.), x_t is the amount of fertilizer applied (lbs./acre), w_t is the wage rate (\$/man-hour), and l_t is the amount of labor employed (man-hours/acre). The per-acre crop yield in year t is a function

$$y_t = f(l_t, x_t, s_t)$$

    of the amount of labor employed and fertilizer applied in year t and the level of fertilizer carryin s_t from the preceding year. Fertilizer carryout in year t is a function

$$s_{t+1} = f(x_t, s_t)$$

    of the amount of fertilizer applied and the level of fertilizer carryin in year t. Assume that future corn prices, fertilizer costs, and wage rates are known with certainty. The corn producer wishes to maximize the expected present value of net revenues over a finite horizon of T years. Formulate the producer's optimization problem. Specifically,

(a) Set up the decision problem (define states, decisions, objective function, transition equation).

(b) Formulate the value function and Bellman's recursive functional equation.

(c) Derive and interpret the first order conditions for optimality.

(d) Assuming an infinite horizon, what conditions characterize the steady-state optimal solution?

15. The role of commodity storage in intertemporal allocation has often been controversial. In particular, the following claims have often been made:

- Competitive storers, in search of speculative profits, tend to hoard a commodity—that is, they collectively store more than is socially optimal.
- A monopolistic storer tends to dump a commodity at first in order to extract monopoly rents in the future—that is, he/she stores less than is socially optimal.

Explore these two propositions in the context of a simple intraseasonal storage model in which a given amount Q of a commodity is to be allocated between two periods. Consumer demand is given by $p_i = a - q_i$ for periods $i = 1, 2$, and the unit cost of storage between periods is k. There is no new production in period 2, so $q_1 + q_2 = Q$. Specifically, answer each of the following:

(a) Determine the amount stored under the assumption that there are a large number of competitive storers.

(b) Determine the amount stored under the assumption that there is a single profit- maximizing storer who owns the entire supply Q at the beginning of period 1.

(c) Taking expected total consumer surplus less storage costs as a measure of societal welfare, determine the socially optimal level of storage. Address the two comments above.

(d) Consider an Economist who rejects net total surplus as a measure of social welfare. Why might he/she still wish to find the level of storage that maximizes total surplus?

To simplify the analysis, assume that the discount factor is 1 and that the storer(s) are risk neutral and possess perfect price foresight.

16. Consider the problem of maximizing the present value of social welfare for an aggregate economy consisting of single composite good. Each year t begins with a predetermined amount of the good s_t, of which an amount c_t is consumed and the remainder x_t is retained as capital. The social welfare derived from consumption in year t is $u(c_t)$ where u is the aggregate utility function. The amount of good available in year $t+1$ is $s_{t+1} = f(x_t)$ where f is the aggregate production function. The utility and production functions exhibit standard curvature properties and the discount rate r is positive.

    (a) How will an increase in the interest rate r affect the long-run levels of consumption and capital stock? Use analytic mathematical methods to structure your argument.

    (b) Suppose now that $s_{t+1} = y_{t+1}f(x_t)$ where y_t is a positive i.i.d. production shock. How must the question in (a) be modified to remain meaningful in a stochastic setting? What techniques would you use to assess, say, whether an increase in the interest rate would raise or lower the long-run variability of consumption.

17. Consider an industry of identical price taking firms. For the representative firm, let s_t denote beginning capital stock, let x_t denote newly purchased capital stock, let $q_t = f(s_t + x_t)$ denote production, let k denote the unit cost of new capital, and let $\gamma > 0$ denote the survival rate of capital. Furthermore, let $p_t = p(q_t)$ be the market clearing price. Find the perfect foresight competitive equilibrium for this industry.

18. Water from a dam can be used for either irrigation or recreation. Irrigation during the spring benefits farmers, but reduces the dam's water level during the summer, damaging recreational users. Specifically, if s_t is the stock of water in the dam at the beginning of year t and an amount x_t is released for irrigation, farmer benefits in year t will be $f(x_t)$ and recreational user benefits will be $u(s_t - x_t)$. Water levels are replenished during the winter months by i.i.d. random rainfalls ϵ_t, giving rise to the water stock transition relationship $s_{t+1} = s_t - x_t + \epsilon_{t+1}$. As a social planner, you wish to find the irrigation policy that max-

imizes the expected discounted sum of farmer and recreational user benefits over an infinite time horizon.

(a) Formulate and interpret Bellman's equation.

(b) Assuming an internal solution, derive and interpret the Euler conditions.

(c) What conditions characterize the certainty equivalent steady-state?

Chapter 9

Discrete Time Continuous State Dynamic Models: Methods

This chapter discusses numerical methods for solving discrete time continuous state dynamic economic models, with emphasis on Markov decision and rational expectations models.

Continuous state dynamic economic models give rise to functional equations whose unknowns are entire functions defined on an interval of Euclidean space. For example, the unknown of a Bellman equation

$$V(s) = \max_{x \in X(s)} \{f(s,x) + \delta E_\epsilon V(g(s,x,\epsilon))\}, \qquad s \in S,$$

is the value function $V(\cdot)$. The unknown of the Euler conditions

$$f_x(s,x(s)) + \delta E_\epsilon \left[\lambda(g(s,x(s),\epsilon)) \cdot g_x(s,x(s),\epsilon)\right] = 0, \qquad s \in S,$$

$$f_s(s,x(s)) + \delta E_\epsilon \left[\lambda(g(s,x(s),\epsilon)) \cdot g_s(s,x(s),\epsilon)\right] = \lambda(s), \qquad s \in S,$$

are the shadow price and policy functions $\lambda(\cdot)$ and $x(\cdot)$. And the unknown of a rational expectations intertemporal equilibrium condition

$$f(s,x(s),Ex(g(s,x(s),\epsilon))), \qquad s \in S,$$

is the response function $x(\cdot)$.

In most applications, the functional equations that arise in dynamic economics lack analytic closed form solution and can only be solved approximately using computational methods. A variety of methods are available for computing approximate solutions to these equations. Linear-quadratic approximation and space discretization historically have been popular among economists. However, in most applications, these methods either provide unacceptably poor approximations or are computationally inefficient.

Only recently have economists begun to employ Galerkin techniques, which have been used among computational physical scientists for decades. Among the various versions of the Galerkin technique, the collocation method is clearly the most useful for solving dynamic models in Economics and Finance. The collocation method is flexible, accurate, and numerically efficient and can be developed directly from basic numerical integration, approximation, and rootfinding methods. Collocation methods may be used to solve discrete and continuous choice Markov decision models and rational expectations models. Bounds and general constraints on variables can also be handled using the method.

The collocation method employs the following general strategy for solving a functional equation:

- Approximate the unknown function with a finite linear combination of n known basis functions whose coefficients are to be determined.
- Require the approximant to satisfy the underlying functional equation at n prescribed points of the domain, called the collocation nodes.

The collocation strategy effectively replaces the functional equation with a finite-dimensional nonlinear equation problem that can be solved using basic nonlinear equation techniques. If the basis functions and nodes are chosen wisely, the collocation method will be numerically consistent; that is, the approximation error can be made arbitrarily small by increasing the number of basis functions and nodes.

The collocation method is a solution strategy rather than a specific technique. When applying the collocation method, the analyst still faces a number of computational modeling decisions. For example, the analyst must choose the basis function and collocation nodes. Numerical approximation theory offers guidance here, suggesting a Chebychev polynomial basis coupled with Chebychev collocation nodes, or a spline basis coupled with equally

spaced nodes will often be good choices. Also, the analyst must chose an algorithm for solving the resulting nonlinear equation. Standard choices include Newton, quasi-Newton, and function iteration methods. A careful analyst will often try a variety of basis-node combinations, and may employ more than one iterative scheme in order to assure the robustness of the results.

Although the collocation method is general in its applicability, the details of implementation vary with the functional equation being solved. Below, the collocation method is developed in greater detail for Bellman equations, Euler conditions, and rational expectations equilibrium conditions.

9.1 Traditional Solution Methods

Before discussing collocation methods for continuous state Markov decision models in greater detail, let us briefly examine the two numerical techniques that historically have been popular among economists for computing approximate solutions to such models: space discretization and linear-quadratic approximation.

Space discretization calls for the continuous state Markov decision problem is to be replaced with a discrete state discrete action Markov decision problem that closely resembles it. To "discretize" the state space of a continuous state Markov decision problem, one partitions the state and action spaces S into finitely many regions, $S_1, S_2, \ldots, S_n$. If the action space X is also continuous, it too is partitioned into finitely many region s $X_1, X_2, \ldots, X_m$. Once the space and action spaces have been partitioned, the analyst selects representative elements, $s_i \in S_i$ and $x_j \in X_j$, from each region. These elements serve as the state and action spaces of the approximating discrete Markov decision problem. The transition probabilities of the discrete problem are computed by integrating with respect to the density of the random shock:

$$P(s_{i'}|s_i, x_j) = \Pr[g(s_i, x_j, \epsilon) \in S_{i'}].$$

If the model is deterministic, then the state is assumed to migrate from state s_i to $s_{i'}$ when decision x_j is taken, if $g(s_i, x_j) \in S_{i'}$.

When the state and action spaces are intervals, say, $S = [s_{min}, s_{max}]$ and $X = [x_{min}, x_{max}]$, it is often easiest to partition the spaces so that the nodes are equally-spaced and the first and final nodes correspond to the endpoints of the intervals. Specifically, we set $s_i = s_{min} + (i-1)w_s$ and $x_j = x_{min} + (j-1)w_x$, for $i = 0, 1, \ldots, n$ and $j = 0, 1, \ldots, m$, where

$w_s = (s_{max} - s_{min})/(n-1)$ and $w_x = (x_{max} - x_{min})/(m-1)$. If the model is stochastic, the transition probabilities of the approximating discrete state decision model are given by

$$P(s_{i'}|s_i, x_j) = \Pr[s_{i'} - w_s/2 \leq g(s_i, x_j, \epsilon) \leq s_{i'} + w_s/2].$$

If the model is deterministic, the state is assumed to migrate from state s_i to $s_{i'}$ when decision x_j is taken, where $s_{i'}$ is the state element nearest $g(s_i, x_j)$.

Another popular method for solving dynamic optimization models is linear-quadratic approximation. Linear-quadratic approximation calls for the transition function g and objective function f to be replaced with linear and quadratic approximants, respectively. Linear-quadratic approximation is motivated by the fact that an unconstrained Markov decision problem with linear transition and quadratic objective has a closed-form solution that is relatively easy to derive. Typically, the linear and quadratic approximants are constructed by forming the first- and second-order Taylor expansions around the certainty-equivalent steady-state. When passing to the linear-quadratic approximation, all inequality constraints of the original problem, including nonnegativity constraints, must be discarded.

The first step in deriving an approximate solution to a continuous state Markov decision problem via linear-quadratic approximation is to compute the certainty-equivalent steady-state. If ϵ^* denotes the mean of ϵ, the certainty-equivalent steady-state state s^*, optimal action x^*, and shadow price λ^* are characterized by the nonlinear equation system:

$$f_x(s^*, x^*) + \delta\lambda^* g_x(s^*, x^*, \epsilon^*) = 0$$

$$\lambda^* = f_s(s^*, x^*) + \delta\lambda^* g_s(s^*, x^*, \epsilon^*)$$

$$s^* = g(s^*, x^*, \epsilon^*).$$

The nonlinear equation may be solved for the steady-state values of s^*, x^*, and λ^* using standard nonlinear equation methods.

The second step is to solve the linear-quadratic control problem whose transition function $\hat{g}$ and objective function $\hat{f}$, are the first- and second-order Taylor series approximants of g and f, respectively:

$$\begin{aligned} f(s,x) \approx \hat{f}(s,x) = f^* + f_s^*(s-s^*) + f_x^*(x-x^*) + 0.5(s-s^*)' f_{ss}^*(s-s^*) \\ +(s-s^*)' f_{sx}^*(x-x^*) + 0.5(x-x^*)' f_{xx}^*(x-x^*) \end{aligned}$$

$$g(s,x,\epsilon) \approx \hat{g}(s,x) = g^* + g_s^*(s-s^*) + g_x^*(x-x^*).$$

Here, f^* and g^* are the values of f and g; f_s^*, f_x^*, g_s^*, and g_x^* are the first partial derivatives of f and g; and f_{ss}^*, f_{sx}^*, and f_{xx}^* are the second partial derivatives of f; all evaluated at the certainty-equivalent steady-state.

The shadow price and optimal policy functions for the linear-quadratic control problem will be linear. Specifically:

$$\lambda(s) = \lambda^* + \Lambda_s(s-s^*)$$

$$x(s) = x^* + X_s(s-s^*)$$

where

$$\begin{aligned}
\Lambda_s &= -[\delta g_s^{*'}\Lambda_s g_x^* + f_{sx}^*][\delta g_x^{*'}\Lambda_s g_x^* + f_{xx}^{*'}]^{-1}[\delta g_x^{*'}\Lambda_s g_s^* + f_{sx}^{*'}] \\
&\quad +\delta g_s^{*'}\Lambda_s g_s^* + f_{ss}^* \\
X_s &= -[\delta g_x^{*'}\Lambda_s g_x^* + f_{xx}^{*'}]^{-1}[\delta g_x^{*'}\Lambda_s g_s^* + f_{sx}^{*'}]
\end{aligned}$$

The first of these two conditions characterizes the slope Λ_s of the approximate shadow price function as a fixed-point of a nonlinear map. The slope can be computed using by either function iteration, typically with initial guess $\Lambda_s = 0$, or by applying the quadratic formula, if the problem is one dimensional. Given the slope Λ_s, the slope X_s of the approximate optimal policy function may be directly computed from the second condition.

If the problem has one dimensional state and action spaces, and if $f_{ss}^* f_{xx}^* = f_{sx}^{*2}$, a condition often encountered in economic problems, then the slope of the shadow price function may be computed analytically as follows:

$$\Lambda_s = [f_{ss}^* g_x^{*2} - 2f_{ss}^* f_{xx}^* g_s^* g_x^* + f_{xx}^* g_s^{*2} - f_{xx}^*/\delta]/g_x^{*2}$$

9.2 Bellman Equation Collocation Methods

Consider Bellman's equation for an infinite horizon discrete time continuous state dynamic decision problem:

$$V(s) = \max_{x \in X(s)} \{f(s,x) + \delta E_\epsilon V(g(s,x,\epsilon))\} \qquad s \in S.$$

To compute an approximate solution to Bellman's equation via collocation, one employs the following strategy: First, one approximates the unknown value function V using a linear combination of known basis functions $\phi_1, \phi_2, \ldots, \phi_n$ whose coefficients $c_1, c_2, \ldots, c_n$ are to be determined:

$$V(s) \approx \sum_{j=1}^{n} c_j \phi_j(s).$$

Second, the basis function coefficients $c_1, c_2, \ldots, c_n$ are fixed by requiring the approximant to satisfy Bellman's equation, not at all possible states, but rather at n states $s_1, s_2, \ldots, s_n$, called the collocation nodes. Many collocation basis-node schemes are available to the analyst, including Chebychev polynomials and nodes, and spline functions and uniform nodes. The best choice of basis-node scheme is application specific, and typically depends on the curvature properties of the value function.

The collocation strategy replaces the Bellman functional equation with a system of n nonlinear equations in n unknowns. Specifically, to compute the approximate solution to the Bellman equation, or more precisely, to compute the n basis coefficients $c_1, c_2, \ldots, c_n$ in the basis representation of the value function approximant, one solves the equation system

$$\sum_j c_j \phi_j(s_i) = \max_{x \in X(s_i)} \{f(s_i, x) + \delta E_\epsilon \sum_{j=1}^{n} c_j \phi_j(g(s, x, \epsilon))\} \qquad i = 1, 2, \ldots, n,$$

which may be compactly expressed in vector form as the *collocation equation*:

$$\Phi c = v(c).$$

Here, Φ, the *collocation matrix*, is the n by n matrix whose typical ij^{th} element is the j^{th} basis function evaluated at the i^{th} collocation node

$$\Phi_{ij} = \phi_j(s_i)$$

and v, the *conditional value function*, is a function from $\Re^n$ to $\Re^n$ whose typical i^{th} element is

$$v_i(c) = \max_{x \in X(s_i)} \{f(s_i, x) + \delta E_\epsilon \sum_{j=1}^{n} c_j \phi_j(g(s_i, x, \epsilon))\}.$$

The conditional value function gives the maximum value obtained when solving the optimization problem embedded in Bellman's equation at each collocation node, given the value function approximation implied by the coefficient vector c.

In principle, the collocation equation may be solved using any nonlinear equation solution method. For example, one may write the collocation equation in the equivalent fixed-point form $c = \Phi^{-1}v(c)$ and use function iteration, which employs the iterative update rule

$$c \leftarrow \Phi^{-1}v(c).$$

Alternatively, one may write the collocation equation as a rootfinding problem $\Phi c - v(c) = 0$ and solve for c using Newton's method, which employs the iterative update rule

$$c \leftarrow c - [\Phi - v'(c)]^{-1}[\Phi c - v(c)].$$

Here, $v'(c)$ is the n by n Jacobian of the conditional value function v at c. The typical element of v' may be computed by applying the Envelope Theorem to the optimization problem that defines $v_i(c)$:

$$v'_{ij}(c) = \frac{\partial v_i}{\partial c_j}(c) = \delta E_\epsilon \phi_j(g(s_i, x_i, \epsilon))$$

where x_i is the optimal argument in the maximization problem in the definition of $v_i(c)$. As a variant to Newton's method one could also employ a quasi-Newton method to solve the collocation equation.

Regardless of which nonlinear equation solution method is used, the conditional value $v_i(c)$ must be computed at every i; that is, the optimization problem embedded in Bellman's equation must be solved at every collocation node s_i, taking the current coefficient vector c as fixed. The Newton method has the additional requirement of computing the Jacobian of v. Computing the Jacobian, however, comes at only a small additional cost because most of the effort required to compute the derivative comes from solving the optimization problem embedded in Bellman's equation, a task that must be performed regardless of the solution method used.

Of course, in any collocation scheme, if the model is stochastic, one must handle the expectation operation in a numerically practical way. Based on numerical analysis theory and practice, a Gaussian quadrature scheme is strongly recommended in collocation strategies where the shock has a conventional continuous distribution. When using a Gaussian quadrature scheme,

the continuous random variable ϵ in the state transition function is replaced with a discrete approximant, say, one that assumes values $\epsilon_1, \epsilon_2, \ldots, \epsilon_m$ with probabilities $w_1, w_2, \ldots, w_m$, respectively. In this instance, the conditional value function v takes the form

$$v_i(c) = \max_{x \in X(s_i)} \{f(s_i, x) + \delta \sum_{k=1}^{m} \sum_{j=1}^{n} w_k c_j \phi_j(g(s_i, x, \epsilon_k))\}.$$

and its Jacobian takes the form

$$v'_{ij}(c) = \delta \sum_{k=1}^{m} w_k \phi_j(g(s_i, x_i, \epsilon_k)).$$

In practice, the critical step in solving Bellman's equation via collocation is coding a numerical routine to evaluate the conditional value function $v(c)$ and its Jacobian. For reasons that will be made clear shortly, one should write the numerical routine so that solves the optimization problem embedded in Bellman's equation, not just at the collocation nodes, but any arbitrary vector of states. More specifically, given an n-degree interpolation scheme selected by the analyst, the routine should solve the optimization problem for every element of an arbitrary m vector s of state nodes and any n-vector c of basis coefficients. The routine should also return the optimal policy at each of the states and the derivative of the values with respect to the basis coefficients.

A Matlab function that performs the necessary optimization when the state and action spaces are one-dimensional and the actions are bounded is given below. Specifically, the function solves the Karush-Kuhn-Tucker conditions of the embedded optimization problem as a complementarity problem:[1]

```
function [v,vc] = vmax(c,s);
[xl,xu] = bfunc(s);
for it=1:maxit
   xold = x;
   [f,fs,fx,fxx] = ffunc(s,x);
   v = f; vx = fx; vxx = fxx;
   for k=1:m
```

[1]The code is abbreviated in that several parameters that must be passed and certain default value computations have been omitted for clarity. For fully functioning code that executes the desired operations, see the Matlab library routine `vmax`.

```
      [g,gs,gx,gxx] = gfunc(s,x,e(k));
      vnval = evalbase(g,n,smin,smax,c,'basecheb');
      vnder = evalbase(g,n,smin,smax,c,'basecheb',1);
      vnsec = evalbase(g,n,smin,smax,c,'basecheb',2);
      v   = v   + delta*w(k)*vnval;
      vx  = vx  + delta*w(k)*vnder.*gx;
      vxx = vxx + delta*w(k)*(vnder.*gxx + vnsec.*gx.*gx);
   end
   x = x - vx./vxx; x = min(x,xu); x = max(x,xl);
   if norm(x-xold,inf)<tol, break, end;
end
vc  = zeros(length(s),n);
for k=1:m
   g = gfunc(s,x,e(k));
   phinxt = basecheb(g,n,smin,smax);
   vc = vc + delta*w(k)*phinxt;
end
```

Here, on input, `s` is an m-vector of states and `c` is an n-vector of basis coefficients of the current value function approximant; and, on output, `v` is the m-vector of optimal values obtained by solving the optimization embedded in Bellman's equation at each state and `vc` is the m-by-n vector of partial derivatives of the values with respect to the basis coefficients.

The function presumes that the analyst has coded routines `ffunc`, `gfunc`, and `bfunc`, designed to compute the reward, transition, and bound functions and their derivatives, respectively, at arbitrary states and actions. It also presumes that the analyst has specified the lower and upper bounds of the state interval, `smin` and `smax`, and the degree of interpolation `n`, and has approximated the shock with a discrete random variable with nodes `e` and weights `w`. The script also presumes that the analyst has chosen a specific interpolation basis, in this case the Chebychev polynomial basis.

Once the maximization routine and reward, transition, and bound function routines have been coded, solution of Bellman's equation via collocation is straightforward. First, the analyst forms the collocation nodes `s` and interpolation matrix `phi`:

```
s   = nodecheb(n,smin,smax).
phi = basecheb(s,n,smin,smax);
```

Given the collocation nodes and collocation matrix, the analyst may then solve the collocation equation via function iteration

```
c = zeros(n,1);
for it=1:maxit
   cold = c;
   v = vmax(c,s);
   c = phi\v;
   if norm(c-cold)<tol, break, end;
end
```

or by Newton iteration

```
c = zeros(n,1);
for it=1:maxit
   cold = c;
   [v,vc] = vmax(c,s);
   c = cold - [phi-vc]\[phi*c-v];
   if norm(c-cold)<tol, break, end;
end
```

Here, `tol` and `maxit` are iteration control parameters set by the analyst and the basis coefficients `c` are initially set to zero, although a better guess may be substituted, if available.

Once convergence apparently has been achieved, the analyst must perform two essential diagnostic checks to assure that the approximate solution is viable. First, since interpolants may provide inaccurate approximations when evaluated outside the interpolation interval, one must check to ensure that states remain within the interpolation interval in all transitions from the collocation nodes. This can be done easily as follows:

```
g = [];
for k=1:m;
  g = [g gfunc(s,x,e(k))];
end
if min(min(g))<smin, disp('Warning: reduce smin'), end;
if max(max(g))>smax, disp('Warning: increase smax'), end;
```

Next, one must check to see that value function approximant solves Bellman's equation to an acceptable degree of accuracy over the entire approximation interval. Since, by construction, the approximant generated by the

solving the collocation equation must solve Bellman's equation exactly at the collocation nodes, this amounts to checking the approximation error at non node points. The easiest way to do this is to plot, over a fine grid spanning the interpolation interval, the residual between the values obtained from the approximant and the values obtained by directly solving the optimization problem embedded in Bellman's equation. For example, approximation residual could be checked at 500 equally spaced nodes as follows:

```
nplot = 500;
splot = nodeunif(nplot,smin,smax);
resid = vmax(c,splot)-evalbase(splot,n,smin,smax,c,'basecheb');
plot(splot,resid)
```

If the residual appears to be reasonably small throughout the entire approximation interval, the computed value function approximant is accepted; otherwise it is rejected and a new approximation is computed using either more collocation nodes or an alternative interpolation scheme. Notice that, to perform this diagnostic, `vmax` is evaluated at states that are not collocation nodes—this is why `vmax` should be constructed to accept an arbitrary vector of states, not just the collocation nodes.

9.3 Euler Equation Collocation Methods

Euler equation methods call for solving the first-order Euler equilibrium conditions of the continuous-space decision problem for the unknown shadow price function λ. Consider the two Euler conditions for an infinite horizon discrete time continuous state dynamic decision problem. The first condition, called the equilibrium condition, derives from the application of the Karush-Kuhn-Tucker Theorem to the optimization problem embedded in Bellman's equation:

$$f_x(s, x(s)) + \delta E_\epsilon \left[\lambda(g(s, x(s), \epsilon)) \cdot g_x(s, x(s), \epsilon)\right] = 0, \qquad s \in S.$$

The second condition, called the Envelope condition, derives from the application of the Envelope Theorem to the optimization problem embedded in Bellman's equation:

$$f_s(s, x(s)) + \delta E_\epsilon \left[\lambda(g(s, x(s), \epsilon)) \cdot g_s(s, x(s), \epsilon)\right] = \lambda(s), \qquad s \in S.$$

To compute an approximate solution to the Euler conditions via collocation, one may employ the following strategy: First, one approximates the unknown shadow price function λ using a linear combination of known basis functions $\phi_1, \phi_2, \ldots, \phi_n$ whose coefficients $c_1, c_2, \ldots, c_n$ are to be determined:

$$\lambda(s) \approx \sum_{j=1}^{n} c_j \phi_j(s).$$

Second, the basis function coefficients $c_1, c_2, \ldots, c_n$ are fixed by requiring the approximant to satisfy the Euler conditions, not at all possible states, but rather at n states $s_1, s_2, \ldots, s_n$, called the collocation nodes.

The collocation strategy replaces the Euler functional equations with a system of n nonlinear equations in n unknowns. Specifically, to compute the approximate solution to the Euler conditions, or more precisely, to compute the n basis coefficients $c_1, c_2, \ldots, c_n$ in the basis representation of the shadow price function approximant, one solves the equation system

$$\sum_{j=1}^{n} c_j \phi_j(s_i) = f_s(s_i, x_i) + \delta E_\epsilon \left[\sum_{j=1}^{n} c_j \phi_j(g(s_i, x_i, \epsilon)) \cdot g_s(s_i, x_i, \epsilon) \right] \qquad i = 1, 2, \ldots, n,$$

where x_i satisfies the equilibrium condition

$$f_x(s_i, x_i) + \delta E_\epsilon \left[\sum_{j=1}^{n} c_j \phi_j(g(s_i, x_i, \epsilon)) \cdot g_x(s_i, x_i, \epsilon) \right] = 0, \qquad i = 1, 2, \ldots, n.$$

This system of equations may be compactly expressed in vector form as the *collocation equation*:

$$\Phi c = p(c).$$

Here, Φ, the *collocation matrix*, is the n by n matrix whose typical ij^{th} element is the j^{th} basis function evaluated at the i^{th} collocation node

$$\Phi_{ij} = \phi_j(s_i)$$

and p, the *conditional shadow price function*, is a function from $\Re^n$ to $\Re^n$ whose typical i^{th} element is

$$p_i(c) = f_s(s_i, x_i) + \delta E_\epsilon \left[\sum_{j=1}^{n} c_j \phi_j(g(s_i, x_i, \epsilon)) \cdot g_s(s_i, x_i, \epsilon) \right] \qquad i = 1, 2, \ldots, n,$$

where x_i solves the equilibrium condition above.

In principle, the collocation equation may be solved using any nonlinear equation solution method. However, due to the complexity of the Jacobian of the conditional shadow price function, derivative-free method are recommended. For example, one may write the collocation equation in the equivalent fixed-point form $c = \Phi^{-1}p(c)$ and use function iteration, which employs the iterative update rule

$$c \leftarrow \Phi^{-1}p(c).$$

Alternatively, one may write the collocation equation as a rootfinding problem $\Phi c - p(c) = 0$ and solve for c using a quasi-Newton method,

Regardless of which nonlinear equation solution method is used, the conditional shadow price $p_i(c)$ must be computed at every i; that is, the equilibrium condition must be solved at every collocation node s_i, taking the current coefficient vector c as fixed. In practice, the critical step in solving the Euler conditions via collocation is coding a numerical routine to solve the Euler equilibrium condition. For reasons that will be made clear shortly, one should write the numerical routine so that it solves the condition, not just at the collocation nodes, but any arbitrary vector of states. More specifically, given an n-degree interpolation scheme selected by the analyst, the routine should solve the equilibrium condition for every element of an arbitrary m vector s of state nodes and any n-vector c of basis coefficients. The routine should also return the optimal policy at each of the states.

A Matlab function that performs the necessary operations when the state and action spaces are one-dimensional and actions are bounded is given below. Specifically, the function solves the equilibrium conditions as a complementarity problem:[2]

```
function p = euler(c,s);
[xl,xu] = bfunc(s);
for it=1:maxit
   xold = x;
   [f,fs,fx,fxx] = ffunc(s,x);
   p = fx; px = fxx;
   for k=1:m
```

[2]The code is abbreviated in that several parameters that must be passed and certain default value computations have been omitted for clarity. For fully functioning code that executes the desired operations, see the Matlab library routine `euler`.

```
      [g,gs,gx,gxx] = gfunc(s,x,e(k));
      pn    = evalbase(g,n,smin,smax,c,'basecheb');
      pnder = evalbase(g,n,smin,smax,c,'basecheb',1);
      p  = p  + delta*w(k)*pn.*gx;
      px = px + delta*w(k)*(pn.*gxx + pnder.*gx.*gx);
   end
   x = x - p./px; x = min(x,xu); x = max(x,xl);
   if norm(x-xold,inf)<tol, break, end;
end
```

Here, on input, `s` is an m-vector of states and `c` is an n-vector of basis coefficients of the current shadow price function approximant; and, on output, `p` is the m-vector of shadow prices obtained by solving the equilibrium conditions at each state.

The function presumes that the analyst has coded routines `ffunc`, `gfunc`, and `bfunc`, designed to compute the reward, transition, and bound functions and their derivatives, respectively, at arbitrary states and actions. It also presumes that the analyst has specified the lower and upper bounds of the state interval, `smin` and `smax`, and the degree of interpolation `n`, and has approximated the shock with a discrete random variable with nodes `e` and weights `w`. The script also presumes that the analyst has chosen a specific interpolation basis, in this example the Chebychev polynomial basis.

Once the maximization routine and reward, transition, and bound function routines have been coded, solution of the Euler conditions via collocation is straightforward. First, the analyst forms the collocation nodes `s` and interpolation matrix `phi`:

```
s   = nodecheb(n,smin,smax).
phi = basecheb(s,n,smin,smax);
```

Given the collocation nodes and collocation matrix, the analyst may then solve the collocation equation via function iteration

```
c = zeros(n,1);
for it=1:maxit
   cold = c;
   p = euler(c,s);
   c = phi\p;
   if norm(c-cold)<tol, break, end;
end
```

Here, `tol` and `maxit` are iteration control parameters set by the analyst and the basis coefficients `c` are initially set to zero, although a better guess may be substituted, if available.

Once convergence apparently has been achieved, the analyst must perform two essential diagnostic checks to assure that the approximate solution is viable. First, since interpolants may provide inaccurate approximations when evaluated outside the interpolation interval, one must check to ensure that states remain within the interpolation interval in all transitions from the collocation nodes. This can be done easily as follows:

```
g = [];
for k=1:m;
  g = [g gfunc(s,x,e(k))];
end
if min(min(g))<smin, disp('Warning: reduce smin'), end;
if max(max(g))>smax, disp('Warning: increase smax'), end;
```

Next, one must check to see that shadow price function approximant solves the Envelope condition to an acceptable degree of accuracy over the entire approximation interval. Since, by construction, the approximant generated by the solving the collocation equation must solve the Envelope condition exactly at the collocation nodes, this amounts to checking the approximation error at non node points. The easiest way to do this is to plot, over a fine grid spanning the interpolation interval, the residual between the shadow prices obtained from the approximant and the shadow prices obtained by directly solving the equilibrium condition. For example, approximation residual could be checked at 500 equally spaced nodes as follows:

```
nplot = 500;
splot = nodeunif(nplot,smin,smax);
resid = euler(c,splot)-evalbase(splot,n,smin,smax,c,'basecheb');
plot(splot,resid)
```

If the residual appears to be reasonably small throughout the entire approximation interval, the computed shadow price function approximant is accepted; otherwise it is rejected and a new approximation is computed using either more collocation nodes or an alternative interpolation scheme. Notice that, to perform this diagnostic, `euler` is evaluated at states that are not collocation nodes—this is why `euler` should be constructed to accept an arbitrary vector of states, not just the collocation nodes.

9.4 Dynamic Programming Examples

9.4.1 Optimal Stopping

The optimal stopping problem is easier to solve numerically than other dynamic optimization problems because for any given state it involves a discrete, binary choice. As such, updating the value function when solving Bellman's equation requires little more than choosing between the maximum of two computed values. In contrast, a dynamic model with continuous action space requires the Karush-Kuhn-Tucker conditions to be solved, or requires the application of some other method of continuous space optimization. Euler condition methods are not applicable to the optimal stopping problem because the Euler conditions are not well-defined when the choice variable is discrete.

To solve the Bellman equation of the optimal stopping problem numerically by collocation, one first uses Gaussian quadrature methods to replace the shock ϵ with a discrete random variable, say, one that assumes values $\epsilon_1, \epsilon_2, \ldots, \epsilon_m$ with probabilities $w_1, w_2, \ldots, w_m$, respectively. If the transition function g is monotonic in ϵ, say, increasing in ϵ, then one can easily compute a minimum and maximum state for the value function interpolation interval by solving the two univariate fixed point problems

$$s_{min} = g(s_{min}, \epsilon_{min})$$

$$s_{max} = g(s_{max}, \epsilon_{max})$$

These two state values define an interval $I = [s_{min}, s_{max}]$ with the property that $g(s, \epsilon_j) \in I$ for all j whenever $s \in I$. That is, given the shock discretization, the interval will not be extrapolated by the numerical collocation routine if the collocation nodes are chosen within the interval.

To compute an approximate solution to Bellman's equation via collocation, one employs the following strategy: One approximates the unknown value function V using a linear combination of known basis functions $\phi_1, \phi_2, \ldots, \phi_n$ defined on I, whose basis coefficients $c_1, c_2, \ldots, c_n$ are to be determined:

$$V(s) \approx \sum_{j=1}^{n} c_j \phi_j(s).$$

One then fixes the basis function coefficients $c_1, c_2, \ldots, c_n$ by requiring the approximant to satisfy Bellman's equation at n collocation nodes $s_1, s_2, \ldots, s_n$ in I. Specifically, one solves the nonlinear vector collocation equation

$$\Phi c = v(c)$$

where Φ is the interpolation matrix associated with the underlying basis-node interpolation scheme and

$$v_i(c) = \max_{x \in X(s_i)} \{f(s_i), \delta \sum_{k=1}^{m} \sum_{j=1}^{n} c_j \phi_j(g(s_i, \epsilon_k))\}.$$

To solve the collocation equation via Newton's method further requires one to compute the Jacobian of v, which is given by

$$v'_{ij}(c) = \frac{\partial v_i}{\partial c_j}(c) = \begin{cases} 0 & v_i(c) > f(s_i) \\ \delta \sum_{k=1}^{m} \phi_j(g(s_i, \epsilon_k)) & \text{otherwise} \end{cases}$$

The Bellman equation for an infinite put option is solved via collocation in the Matlab demo file demo8001.m. The infinite put option is a contract that allows the bearer to sell a given quantity of a commodity or financial asset to the writer of the option at a specified strike price at any given future date. When exercised, the value of the put option is the difference between the strike price and the current market price, whenever the former is the larger of the two; the put option provides no reward to the bearer if exercised when the market price exceeds the strike price.

In demo0801.m, the strike price is assumed to be 1 and the underlying commodity price is assumed to follow a simple first order autoregressive process

$$\log p_{t+1} = \gamma * log(p_t) + \epsilon_t$$

where $\gamma = 0.8$ and ϵ_t is i.i.d. normal with zero mean and variance $\sigma^2 = 0.2$. The key to solving the Bellman equation for the optimal value and policy function is implementing the routine that evaluates the value of exercising and holding the option for any given current price p:

```
function [v,vc] = vmax(c,p,basepass);
vexer = max(pstrk-exp(p),0);
vkeep = zeros(size(p));
for k=1:m
  pnext = pbar + gamma*(p-pbar) + e(k);
```

```
  vnext = evalbase(pnext,n,pmin,pmax,c,basepass);
  vkeep = vkeep + delta*w(k)*vnext;
end
v = max(vexer,vkeep);
vc  = zeros(length(p),n);
for k=1:m
  pnext = pbar + gamma*(p-pbar) + e(k);
  phinxt = eval([basepass,'(pnext,n,pmin,pmax)']);
  vc = vc + delta*w(k)*phinxt;
end
for i=1:length(p)
   if vexer(i)>vkeep(i), vc(i,:) = 0; end;
end
```

Here, `p` refers to a vector of current prices, `pstrik` refers to the strike price, and `pmin` and `pmax` refer to the minimum and maximum prices achievable with the discretized normal innovation.

9.4.2 Stochastic Optimal Growth

Consider the problem of numerically solving the stochastic optimal growth problem of the preceding chapter under the assumption that $u(c) = c^{1-\alpha}/(1-\alpha)$, $f(x) = x^{\beta}$, and $\log(\epsilon)$ is i.i.d Normal$(0, \sigma^2)$.

To solve the growth model by linear-quadratic approximation one first computes the certainty-equivalent steady-state action, state, and shadow price in sequence:

$$x^* = \left(\frac{1-\delta\gamma}{\delta\beta}\right)^{\frac{1}{\beta-1}}$$

$$s^* = \gamma x^* + x^{*\beta}$$

$$\lambda^* = (s^* - x^*)^{-\alpha}.$$

Using the results of section 9.1, it follows that the shadow price and optimal policy function approximant are:

$$\lambda(s) = \lambda^* + \lambda'(s - s^*)$$

$$x(s) = x^* + x'(s - s^*).$$

where

$$\lambda' = -(1-\delta)\alpha(s^* - x^*)^{-\alpha-1}$$

$$x' = \delta.$$

To solve the Bellman equation of the optimal growth model using collocation, one first selects a series of n basis functions ϕ_j and n collocation nodes s_i, and writes the approximation

$$V(s) \approx \sum_{j=1}^{n} c_j \phi_j(s).$$

One also employs a Gaussian quadrature scheme to replace the stochastic shock with a discrete approximant, say, one that assumes values $\epsilon_1, \epsilon_2, \ldots, \epsilon_m$ with probabilities $w_1, w_2, \ldots, w_m$, respectively.

The unknown vector of basis coefficients c is then computed by solving the collocation equation

$$\Phi c = v(c)$$

where Φ is the interpolation matrix constructed by evaluating the basis functions at the collocation nodes and

$$v_i(c) = \max_{0 \le x \le s_i} \{(s_i - x)^{1-\alpha}/(1-\alpha) + \delta E_\epsilon \sum_{k=1}^{m} \sum_{j=1}^{n} w_k c_j \phi_j(\gamma x + \epsilon_k x^\beta)\}.$$

To solve the collocation equation via Newton's method further requires one to compute the Jacobian of v, which is given by

$$v'_{ij}(c) = \frac{\partial v_i}{\partial c_j}(c) = \delta \sum_{k=1}^{m} w_k \phi_j(\gamma x_i + \epsilon_k x_i^\beta)$$

where x_i solves the optimization problem above.

To solve the Euler conditions of the optimal growth model using collocation, one first selects a series of n basis functions ϕ_j and n collocation nodes s_i, and writes the approximation

$$\lambda(s) \approx \sum_{j=1}^{n} c_j \phi_j(s).$$

As with the Bellman equation, on also employs a Gaussian quadrature scheme to replace the stochastic shock with a discrete approximant.

The unknown vector of basis coefficients c is then computed by solving the collocation equation

$$\Phi c = p(c)$$

where Φ is the interpolation matrix constructed by evaluating the basis functions at the collocation nodes and

$$p_i(c) = (s_i - x_i)^{-\alpha}$$

where x_i solves the equilibrium condition

$$-(s_i - x_i)^{-\alpha} + \delta \sum_{k=1}^{m} w_k \left[\hat{\lambda}(\gamma x_i + \epsilon_k x_i^{\beta}) \cdot [\gamma + \epsilon_k \beta x_i^{\beta-1}] \right] = 0.$$

A demonstration program, demo0802.m, provided with the Matlab library accompanying the lecture notes solves the optimal growth model under the assumptions that $\alpha = 0.2$, $\beta = 0.5$, $\gamma = 0.9$, and $\delta = 0.9$. Figures (*) and (*) give approximate optimal capital retention and shadow price functions derived using different approximation methods to solve the Bellman equation. In figure (*) the Chebychev polynomial approximant exhibits the characteristic smoothness of the true optimal solution. The relation is nearly linear; even so, the linear approximant obtained through linear-quadratic approximation has the wrong slope and can produce large errors away from the steady-state. The discrete-space approximant tends to follow the Chebychev approximant, but can exhibit large errors locally. Finally, in figure (*), the linear-quadratic shadow price function approximant can yield extremely large errors, particularly because the true relation is nonlinear.

9.4.3 Renewable Resource Problem

Consider the renewable resource problem under the assumptions that $p(x) = x^{-\gamma}$, $c(x) = kx$, and $g(s, x) = \alpha(s - x) - 0.5\beta(s - x)^2$.

To solve the renewable resource model by linear quadratic approximation one first computes the certainty equivalent steady state state, action, and shadow price in sequence:

$$s^* = \frac{\alpha^2 - \delta^{-2}}{2\beta}$$

$$x^* = s^* - \frac{\delta\alpha - 1}{\delta\beta}$$

$$\lambda^* = (x^*)^{-\gamma} - k.$$

Using the results above, it then follows that the shadow price and optimal policy function approximant are:

$$\lambda(s) = \lambda^* - \frac{1-\delta}{\gamma(x^*)^{1+\gamma}}(s - s^*)$$

$$x(s) = x^* + (1-\delta)(s - s^*).$$

To solve the Bellman equation of the renewable resource model using collocation, one first selects a series of n basis functions ϕ_j and n collocation nodes s_i, and writes the approximation

$$V(s) \approx \sum_{j=1}^{n} c_j \phi_j(s).$$

The unknown vector of basis coefficients c is then computed by solving the collocation equation

$$\Phi c = v(c)$$

where Φ is the interpolation matrix constructed by evaluating the basis functions at the collocation nodes and

$$v_i(c) = \max_{0 \le x \le s_i} \{x^{1-\gamma}/(1-\gamma) - kx + \delta \sum_{j=1}^{n} c_j \phi_j(\alpha(s_i - x) - 0.5\beta(s_i - x)^2)\}.$$

To solve the collocation equation via Newton's method further requires one to compute the Jacobian of v, which is given by

$$v'_{ij}(c) = \frac{\partial v_i}{\partial c_j}(c) = \delta\phi_j(\alpha(s_i - x_i) - 0.5\beta(s_i - x_i)^2)$$

where x_i solves the maximization problem above.

To solve the Euler conditions of the renewable resource model using collocation, one first selects a series of n basis functions ϕ_j and n collocation nodes s_i, and writes the approximation

$$\lambda(s) \approx \sum_{j=1}^{n} c_j \phi_j(s).$$

The unknown vector of basis coefficients c is then computed by solving the collocation equation

$$\Phi c = p(c)$$

where Φ is the interpolation matrix constructed by evaluating the basis functions at the collocation nodes and

$$p_i(c) = \delta\lambda(\alpha(s_i - x_i) - 0.5\beta(s_i - x_i)^2) \cdot [\alpha - \beta(s_i - x_i)]$$

where x_i solves the equilibrium condition

$$x_i^{-\gamma} - k - \delta\lambda(\alpha(s_i - x_i) - 0.5\beta(s_i - x_i)^2) \cdot [\alpha - \beta(s_i - x_i)] = 0$$

A demonstration program, demo0803.m, provided with the Matlab library accompanying the lecture notes solves the renewable resource model under the assumptions that $\gamma = 0.5$, $\alpha = 4$, $\beta = 1$, $k = 0.2$, and $\delta = 0.9$. Figures (*) and (*) give approximate optimal harvest and shadow price functions derived using different approximation methods to solve the Bellman equation. In figure (*) the Chebychev polynomial approximant exhibits the smoothness of the true optimal solution. Again, the linear-quadratic approximant has the wrong slope and can produce large errors away from the steady-state. Finally, in figure (*) the linear-quadratic shadow price function approximant can yield extremely large errors away from the steady-state because the true relation is nonlinear.

9.4.4 Nonrenewable Resource Problem

Consider the nonrenewable resource problem under the assumption that the cost of extraction is $c(s, x) = x^2/(s + \beta)$.

To solve the Bellman equation of the nonrenewable resource model using collocation, one first selects a series of n basis functions ϕ_j and n collocation nodes s_i, and writes the approximation

$$V(s) \approx \sum_{j=1}^{n} c_j \phi_j(s).$$

The unknown vector of basis coefficients c is then computed by solving the collocation equation

$$\Phi c = v(c)$$

where Φ is the interpolation matrix constructed by evaluating the basis functions at the collocation nodes and

$$v_i(c) = \max_{0 \le x \le s_i} \{\alpha x - x^2/(s+\beta) + \delta \sum_{j=1}^{n} c_j \phi_j(s_i - x)\}.$$

To solve the collocation equation via Newton's method further requires one to compute the Jacobian of v, which is given by

$$v'_{ij}(c) = \frac{\partial v_i}{\partial c_j}(c) = \delta \phi_j(s_i - x_i)\}$$

where x_i solves the maximization problem above.

To solve the Euler conditions of the nonrenewable resource model using collocation, one first selects a series of n basis functions ϕ_j and n collocation nodes s_i, and writes the approximation

$$\lambda(s) \approx \sum_{j=1}^{n} c_j \phi_j(s).$$

The unknown vector of basis coefficients c is then computed by solving the collocation equation

$$\Phi c = p(c)$$

where Φ is the interpolation matrix constructed by evaluating the basis functions at the collocation nodes and

$$p_i(c) = x_i^2/(s+\beta)^2 + \delta \sum_{j=1}^{n} c_j \phi_j(s_i - x_i)$$

where x_i solves the equilibrium condition

$$\alpha - 2x/(s+\beta) + -\delta \sum_{j=1}^{n} c_j \phi_j(s_i - x) = 0.$$

A demonstration program, demo0803.m, provided with the Matlab library accompanying the lecture notes solves the nonrenewable resource model under the assumptions that $\alpha = 1$, $\beta = 10$, and $\delta = 0.9$. Figures * and * give approximate optimal extraction and shadow price functions derived using different approximation methods. In figure 6.3a, the Chebychev polynomial

approximant exhibits the essential properties of the true optimal solution. As can be seen in figure *, the optimal policy for stock levels less than 0.5 is to extract the entire stock or ore. In other words, over this range, the constraint $x \leq s$ is binding. As see in figure *, the slope of the linear-quadratic approximant for the shadow price function has the wrong sign. This is due to the fact that in linear-quadratic approximation we ignore an essential constraint.

9.5 Rational Expectation Collocation Methods

I did not get to cover these methods in sufficient depth in class. I will not exam you on them.

9.5.1 Example: Asset Pricing Model

9.5.2 Example: Commodity Storage

Consider the commodity storage problem of the preceding chapter under the assumptions that

- $p(s-x) = (s-x)^{-\gamma}$
- $c(x) = \alpha + \beta \log(x)$
- $\log(h_t)$ is i.i.d Normal$(0, \sigma^2)$

where γ, α, and β are positive constants.

If $\lambda(s)$ is the commodity price given supply s, then the equilibrium storage level x satisfies

$$-(s-x)^{-\gamma} - \alpha - \beta \log(x) + \delta E\lambda(x+h) = 0$$

$$\lambda(s) = (s-x)^{-\gamma}.$$

To solve the storage model by linear-quadratic approximation one first computes the certainty-equivalent steady-state price, action, and state in sequence:

$$\lambda^* = 1$$

$$x^* = \exp\left(\frac{\delta - 1 - \alpha}{\beta}\right)$$

$$s^* = x^* + 1.$$

Using the results of the preceding chapter, it follows that the shadow price and optimal policy function approximant are:

$$\lambda(s) = \lambda^* + \lambda'(s - s^*)$$

$$x(s) = x^* + x'(s - s^*).$$

where

$$\lambda' = -\gamma - \gamma^2/[\delta\lambda' - \gamma - \beta/x^*]$$

$$x' = -\gamma/[\delta\lambda' - \gamma - \beta/x^*].$$

The value of λ' can be computed by successive approximation; given λ', the value of x' is easily computed.

To solve the rational expectations equilibrium conditions of the storage model using collocation, one first selects a series of n basis functions ϕ_j and n collocation nodes s_i, and writes the approximation to the equilibrium price function

$$\lambda(s) \approx \sum_{j=1}^{n} c_j \phi_j(s).$$

The unknown vector of basis coefficients c is then computed by solving the collocation equation

$$\Phi c = p(c)$$

where Φ is the interpolation matrix constructed by evaluating the basis functions at the collocation nodes and

$$p_i(c) = (s_i - x_i)^{-\gamma}$$

where x_i solves the equilibrium condition

$$-(s_i - x_i)^{-\gamma} - \alpha - \beta \log(x_i) + \delta \sum_{j=1}^{n} \sum_{k=1}^{m} w_k c_j \phi_j(x_i + h_k) = 0.$$

The Matlab demonstration file demo8022.m solves the commodity rational expectations model under the assumptions that $\gamma = 5.0$, $\alpha = 0.6$, $\beta = 0.1$, $\sigma = 0.15$, and $\delta = 0.9$. For this parameterization, $s^* = 1.001$, $x^* = 0.001$, $\lambda^* = 1.0$, $\lambda' = -4.7716$, and $x' = 0.0457$. Figures (*) and (*) give approximate rational expectations equilibrium storage and price functions derived using different approximation methods. In figure (*), the Chebychev polynomial approximant exhibits the essential properties of the true rational expectations equilibrium solution. As can be seen in figure (*), prices rise and storage drops to near zero when supplies are short. As supply rises, however, prices drop and stockholding becomes profitable. As see in figure (*) and (*), the solution to the linearized rational expectations model can give misleading results. In particular, the linear model allows for negative stockholding and negative prices, neither of which are observed in practice.

9.6 Comparison of Solution Methods

In developing a numerical approximation strategy for solving Bellman's equation, one pursues a series of multiple, sometimes conflicting goals. First, the algorithm should offer a high degree of accuracy for a minimal computational effort. Second, the algorithm should be capable of yielding arbitrary accuracy, given sufficient computational effort. Third, the algorithm should yield answers with minimal convergence problems. Fourth, it should be possible to code the algorithm relatively quickly with limited chances for programmer error.

Space discretization has some major advantages for computing approximate solutions to continuous-space dynamic decision problems. The biggest advantage to space discretization is that it is easy to implement. In particular, the optimization problem embedded in Bellman's equation is solved by complete enumeration, which is easy to code and numerically stable. Also, constraints are easily handled by the complete enumeration algorithm. Each time a new action is examined, one simply tests whether the action satisfies the constraint, and rejects it if it fails to do so. Finally, space discretization can provide an arbitrarily accurate approximation by increasing the number of state nodes.

Space discretization, however, has several major disadvantages. The biggest disadvantage is that complete enumeration is extremely slow. Complete enumeration mindlessly examines all possible actions, ignoring the

derivative information that would otherwise help to find the optimal action. Another drawback to space discretization is that it uses discontinuous step functions to approximate the value and policy functions. The approximate optimal solution generated by space discretization will not possess the smoothness and curvature properties of the true optimal solution. Finally, because the states and actions are forced to coincide with specified nodes, the accuracy afforded by space discretization will be limited by the coarseness of the state and action space grids.

Linear-quadratic approximation is perhaps the method easiest to implement. The solution to the approximating problem is a linear function whose coefficients can be derived analytically using the methods discussed in section (*). Alternatively, the coefficients can easily be computed numerically using a successive approximation scheme that is typically free of convergence problems.

Linear-quadratic approximation, however, has some severe shortcomings. The basic problem with linear-quadratic approximation is that it relies on Taylor series approximations that are accurate only in the vicinity of the steady-state, and then only if the process is deterministic or nearly so. Linear-quadratic approximation will yield poor results if random shocks repeatedly throw the state variable far from the steady-state and if the reward and state transition functions are not accurately approximated by second- and first-degree polynomials over their entire domains. Linear-quadratic approximation will yield especially poor approximations if the true optimal process is likely to encounter any inequality and nonnegativity constraints, which must be discarded in passing to a linear-quadratic approximation.

Collocation methods address many of the shortcomings of linear-quadratic approximation and space discretization methods. Unlike linear-quadratic approximation, collocation methods employ global, rather than local, function approximation schemes and, unlike space discretization, they approximate the solution using a smooth, not discontinuous, function. Chebychev collocation methods, in particular, are motivated by the Wieirstrass polynomial approximation theorem, which asserts that a smooth function can be approximated to any level of accuracy using a polynomial of sufficiently high degree. A second important advantage to collocation methods is that they may employ rootfinding or optimization that exploit derivative information. A differentiable approach can help pinpoint the equilibrium solution at each state node faster and more accurately than the complete enumeration scheme of discrete dynamic programming.

The collocation method replaces the inherently infinite-dimensional functional equation problem with a finite-dimensional nonlinear equation problem that can be solved using standard nonlinear equation methods. The accuracy afforded by the computed approximant will depend on a number of factors, most notably the number of basis functions and collocation nodes n. The greater the degree of approximation n, the more accurate the resulting approximant, but the more expensive is its computation. For this reason choosing a good set of basis functions and collocation nodes is critical for achieving computational efficiency. Approximation theory suggests that Chebychev polynomials basis functions and Chebychev collocation points will often make superior choices, provided the solution to the functional equation is relatively smooth. Otherwise, linear or cubic basic splines with equally spaced collocation nodes may provide better approximation.

In using collocation schemes, one might be tempted to choose equally spaced points and to represent the interpolating polynomial as the linear combination of the standard monomials. However, as seen in Chapter 3, uniform node polynomial interpolation can yield extremely poor global approximations and can produce explosive approximation error. Also, computing the monomial coefficients of an interpolating polynomial is an ill-conditioned process that is highly vulnerable to rounding error and convergence failure.

Numerical analysis theory suggest that the Chebychev interpolation nodes and Chebychev polynomials are nearly optimal choices for forming polynomial interpolants. Accuracy and efficiency with Chebychev nodes and polynomials are guaranteed by Chebychev polynomial approximation theorem, which asserts that, for a given degree, the best approximating polynomial is the one that interpolates the function at the Chebychev nodes. The theorem also asserts that such approximation error will tend to disappear if the degree of approximation is increased. Also, using this combination of nodes and basis polynomials will ensure that the interpolating matrix will be orthogonal. Thus, computing the coefficients c_j of the interpolating polynomial will be faster and numerically more stable than for other polynomial bases.

Chebychev collocation, however, is not without its disadvantages. First, polynomial interpolants can behave strangely outside the range of interpolation and should be extrapolated with extreme caution. Even when state variable bounds for the model solution are known, states outside the bounds can easily be generated in the early stages of the solution algorithm, leading to convergence problems. Also, polynomial interpolants can behave strangely in the vicinity of nondifferentiabilities in the function being interpolated. In

particular, interpolating polynomials can fail to preserve monotonicity properties near such points, undermining the rootfinding algorithm used to compute the equilibrium at each state node. Finally, inequality constraints, such as nonnegativity constraints, require the use of special methods for solving nonlinear complementarity problems.

Table 1 gives the execution time and approximation error associated with four solution schemes, including uniform polynomial and Chebychev collocation, as applied to the commodity storage model examined in section (*). Approximation error is defined as the maximum absolute difference between the "true" price function and the approximant at points spaced 0.001 units apart over the approximation interval $[0.5, 2.0]$. Execution times are based on the successive approximation algorithm implemented on an 80486 50 megahertz Gateway 2000 personal microcomputer.

The superiority of the Chebychev collocation for solving the storage model is evident from table 1. The accuracy afforded by Chebychev collocation exceeded that of space discretization by several orders of magnitude. For example, the accuracy achieved by space discretization in nearly five minutes of computation was easily achieved by Chebychev collocation in less than one-tenth of a second. In the same amount of time, the linear-quadratic approximation method afforded an approximation that was three orders of magnitude worse than that afforded by Chebychev collocation. The approximation afforded by linear-quadratic approximation, moreover, was not subject to improvement by raising the degree of the approximation, which is fixed. Finally, as seen in table 1, when using uniform node, monomial collocation, the approximation error actually increased as the number of nodes doubled from 10 to 20; the algorithm, moreover, would not converge for more than 23 nodes. The example thus illustrates once again the inconsistency and instability of uniform node monomial interpolation.

9.7 Dynamic Analysis

Although the optimal policy and shadow price functions reveal a great deal about the nature of the optimized dynamic process, they give an incomplete picture of the model's implications. Given an economic model, we typically wish to describe the dynamic behavior of the optimized process, and how its behavior changes with variations model parameters or assumptions. Given a dynamic economic model, we typically characterize the model's solution

Method	Number of Nodes	Execution Time (seconds)	Maximum Absolute Error
Chebychev	10	0.1	4.7E−02
Polynomial	20	0.4	1.1E−02
Collocation	30	0.7	2.7E−03
	40	1.1	5.9E−04
	50	1.6	3.3E−04
	100	5.8	3.1E−06
	150	12.5	2.3E−08
Uniform	10	0.1	1.4E−01
Polynomial	20	0.3	1.7E+00
Collocation	30	N.A.	N.A.
Space	10	2.0	4.5E+00
Discretization	20	7.5	1.7E+00
	30	16.9	8.6E−01
	40	31.0	5.3E−01
	50	32.3	3.5E−01
	100	124.6	9.7E−02
	150	292.2	4.5E−02
L-Q Approximation		0.1	2.8E+01

Table 9.1: Execution Times and Approximation Error for Selected Continuous-Space Approximation Methods

in one of two ways. Steady-state analysis examines the long-run tendencies of the optimized process, abstracting from the initial state and path taken by the process over time. Dynamic path analysis focuses on how the system evolves over time, starting from a given initial condition.

Given a deterministic dynamic model, steady-state and dynamic path analysis are relatively straightforward to perform. As we have seen, the steady-state of a deterministic process is typically characterized by a system of nonlinear equations. The system can be solved numerically and totally differentiated to generate explicit expressions describing how the steady-state varies with changes in model parameters. Dynamic path analysis can be performed through a simple deterministic simulation of the process, which requires repeated evaluations of the optimal policy and state transition functions. In particular, if $x(s)$ is the optimal policy function and $g(s, x)$ is the transition function, then, given an initial state s_0, the path taken by the state variable may be computed recursively as follows: $s_{t+1} = g(s_t, x(s_t))$. Given the path of the state variable s_t, it is then straightforward to generate the path taken by any other endogenous variable.

The analysis of stochastic models is a bit more involved. Stochastic models do not generate an unique, deterministic path from a given initial state. A stochastic process may take any one of many possible paths, depending on the realizations of the random shocks. Often, it is instructive to generate one such possible path to illustrate the volatility that an optimized process is capable of exhibiting. This is performed by a simple Monte Carlo simulation in which a sequence of pseudorandom shocks are generated for the process using a random number generator. In particular, given the optimal policy function $x(s)$, the transition function $g(s, x, \epsilon)$, an initial state s_0, and a pseudorandom sequence of ϵ_t, a representative path may be generated recursively as follows: $s_{t+1} = g(s_t, x(s_t), \epsilon_{t+1})$.

Figure 5 illustrates the difference between the paths taken by deterministic and stochastic models. The paths coincide with the stochastic and deterministic versions of the optimal growth problem of section 6.4. As can be seen in figure 5, the deterministic path is smooth and eventually converges to a steady-state. In contrast, the stochastic path is erratic, reflecting the influences of random production shocks, and does not converge to a set value.

A more revealing analysis of the dynamics generated by a stochastic model is to draw not a single representative path, but rather the expected path of the process. The expected path may be computed by generating a large number of independent representative paths and averaging the results at each

point in time. As seen in figure 6, the expected path exhibits many of the properties of a deterministic path. Specifically, the expected path is smooth and converges to a steady-state. The expected path of the stochastic model, however, should not be confused with the path of the certainty equivalent model. As seen in figure 6, the certainty equivalent model underpredicts the stock level at every point in time and thus in the steady-state.

The steady-state of a stochastic process is a distribution, not a point. Typically, it will suffice to compute the mean and standard deviation of the steady-state distribution for selected endogenous variables. The most common approach to computing steady-state means and variances is through the use of Monte Carlo simulation. Monte Carlo simulation is used to generate a single representative path of long horizon, say 10,000 periods. The values of the endogenous variable thus generated collectively reflect the steady-state distribution of the variable. In practice, we simply accumulate the first and second moments of the variable with each simulated period, and compute the means and the standard deviation at the conclusion of the simulated long-run history.

In many instances we are interested in seeing how certain properties of the model vary as the parameters of the model change. Typically, we focus on the relationship between the steady-state mean or variance of a given endogenous variable and an exogenous parameter of interest. In order to perform sensitivity analysis, one performs Monte Carlo simulations at chosen values of the parameter and constructs a least-squares fit to the graph points generated in this fashion. Figures 7 illustrates this technique. Here, we simulated the storage model at equally-space points for the base storage cost α and drew the relationship between the steady-state standard deviation of price and the storage cost. The figure indicates that increasing the storage costs tends to destabilize price.

Another approach to performing steady-state and path analysis is to convert the continuous-space stochastic process into a discrete one and use Markov chain methods to approximate the expected path and the steady-state distribution of the process.

Chapter 10

Continuous Time Mathematics

10.1 Introduction

In recent years the use of continuous time approaches has become increasingly popular in economics applications, especially in finance, macro and resource economics. Although many models can be implemented in either discrete or continuous time, a major advantage of continuous time arises in modeling intertemporal arbitrage conditions. The essence of intertemporal arbitrage is the construction of portfolios of goods that are risk free and, as such, earn the risk free rate of return in equilibrium. It is generally not possible to construct such risk free portfolios in discrete time (it would take an uncountably infinite number of assets to make a portfolio risk free when the number of possible states of nature is uncountably infinite). Risk free portfolios can be easily constructed in continuous time with a small number of assets so long as the portfolios can be continuously adjusted at zero cost. This leads to an important method of evaluating assets and of determining optimal strategies that complements the dynamic optimization approach (see Pindyck for a discussion of the relationship between these two approaches).

The basic tools used in the analysis of continuous time models are Ito calculus and stochastic control, the latter term referring to dynamic programming in continuous time. Ito processes (defined below) as generally used because they are both flexible and can be handled with relative ease. Although initially the use of Ito calculus requires some mental investment, its use in practice turns out to be nearly as straightforward as calculus applied to deterministic functions.

For many problems of economic interest the boundary conditions present special difficulties. Although economic variables are often bounded below by zero, they typically have no natural upper boundaries. Furthermore, many problems exhibit so-called free boundaries, with one or more differential equations describing the behavior of the variable in regions of the state space with endogenously determined boundaries (examples are discussed in Section 11.2).

In what follows, a brief introduction to practical aspects of Ito processes and Ito's Lemma is presented. Also discussed is a version of the Feynman-Kac equation, which describes an equivalence relationship between the expectation of a functional of an Ito process and the solution of an associated partial differential equation. The section includes a discussion of the use of intertemporal arbitrage to value derivative assets. The concluding section discusses the analysis of transition and long-run (steady-state) probability distributions associated with Ito processes.

Stochastic control techniques are discussed in the next chapter, which focuses on the continuous time Bellman's equation and provides numerous with examples.

10.1.1 Stochastic Models with Ito Processes

The stochastic processes most commonly used in economic applications are constructed from the so-called standard Weiner process or standard Brownian motion. This process is most intuitively defined as a limit of sums of independent normally distributed random variables:

$$z_{t+\Delta t} - z_t \equiv \int_t^{t+\Delta t} dz = \lim_{n\to\infty} \sqrt{\frac{\Delta t}{n}} \sum_{i=1}^{n} v_i.$$

where the v_i are independently and identically distributed standard normal variates ($i.i.d.\ N(0,1)$). The standard Weiner process has the following properties:

1. time paths are continuous (no jumps)
2. non-overlapping increments are independent
3. increments are normally distributed with mean zero and variance Δt.

The first property is not obvious but properties 2 and 3 follow directly from the definition of the process. Each non-overlapping increment of the process is defined as the sum of independent random variables and hence the increments are independent. Each of the variables in the sum have expectation zero and hence so does the sum. The variance is

$$E\Delta z^2 = \Delta t \lim_{n\to\infty} \frac{1}{n} E\left(\sum_{i=1}^{n} v_i\right)^2 = \Delta t \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} E[v_i^2] = \Delta t.$$

Ito diffusion processes are typically represented in differential form as

$$dx = \mu(x,t)dt + \sigma(x,t)dz$$

where z is a standard Wiener process.[1] The Ito process in completely defined in terms of the functions μ and σ, which can be interpreted as the instantaneous mean and standard deviation of the process:

$$E[dx] = \mu(x,t)dt$$

and

$$Var[dx] = E[dx^2] - (E[dx])^2 = E[dx^2] - \mu(x,t)^2 dt^2 = E[dx^2] = \sigma^2(x,t)dt,$$

which are also known as the drift and diffusion terms, respectively. This is not as limiting as it might appear at first, because a wide variety of stochastic behavior can be represented by appropriate redefinition of the two functions.

The differential representation is a shorthand for the stochastic integral

$$x_{t+\Delta t} = x_t + \int_t^{t+\Delta t} \mu(x_\tau, \tau)d\tau + \int_t^{t+\Delta t} \sigma(x_\tau, \tau)dz. \tag{10.1}$$

[1] Standard regularity conditions placed on Ito processes involve restrictions on the μ and σ functions. These include Borel measurability, a Lipschitz condition that for all x and y there exists a k such that

$$\|\mu(x,t) - \mu(y,t)\| + \|\sigma(x,t) - \sigma(y,t)\| \leq k\|x - y\|,$$

and a growth condition that for all x and $t \geq 0$ there is a k such that

$$\|\mu(x,t)\| + \|\sigma(x,t)\| \leq k(1 + \|x\|).$$

These are sufficient conditions to ensure that x is uniquely defined Markov process. Furthermore, the continuity of μ and σ ensures that x is a diffusion.

The first of the integrals in 10.1 is an ordinary (Riemann) integral. The second integral, however, involves the stochastic term dz and requires additional explanation. It is defined in the following way:

$$\int_t^{t+\Delta t} \sigma(x_\tau, \tau) dz = \lim_{n\to\infty} \sqrt{\frac{\Delta t}{n}} \sum_{i=0}^{n-1} \sigma(x_{t+ih}, t+ih) v_i, \tag{10.2}$$

where $h = \Delta t/n$ and $v_i \sim$i.i.d. N(0,1). The key feature of this definition is that it is non-anticipating; values of x that are not yet realized are not used to evaluate the σ function. This naturally represents the notion that current events cannot be functions of specific realizations of future events.[2] It is useful to note that $E_t dx = \mu(x,t)dt$; this is a direct consequence of the fact that each of the elements of the sum in (10.2) has zero expectation. This implies that

$$E_t[x_{t+\Delta t}] = x_t + E_t \int_t^{t+\Delta t} \mu(x_\tau, \tau) d\tau$$

From a practical point of view, the definition of an Ito process as the limit of a sum provides a natural method for simulating discrete realizations of the process using

$$x_{t+\Delta t} = x_t + \mu(x_t, t)\Delta t + \sigma(x_t, t)\sqrt{\Delta t}\, v,$$

where $v \sim N(0,1)$. This approximation will be exact when μ and σ are constants.[3] In other cases the approximation will improve as Δt gets small, but make produce inaccurate results as Δt gets large.

[2]Standard Riemann integrals of continuous functions are defined as:

$$\int_a^b f(x)dx = \lim_{n\to\infty} h \sum_{i=0}^{n-1} f(a + (i+\lambda)h),$$

with $h = (b-a)/n$ and λ is any value on $[0,1]$. With stochastic integrals, alternative values on λ produce different results. Furthermore, any value of λ other than 0 would imply a sort of clairvoyance that makes it unsuitable for applications involving decision making under uncertainty.

[3]When μ and σ are constants the process is known as absolute Brownian motion. An exact simulation method also exists when the drift and diffusion terms are proportional to x. This is the so-called geometric Brownian motion process:

$$dx = \mu x dt + \sigma x dz.$$

In order to define and work with functions of Ito processes it is necessary to have a calculus that operates consistently with them. Suppose $y = f(x,t)$, with continuous derivatives f_x, f_t and f_{xx}. In the simplest case x, z, and y are all scalar processes. It is intuitively reasonable to define the differential dy as

$$dy = f_t dt + f_x dx,$$

as would be the case in deterministic calculus. Unfortunately, this will produce incorrect results because it ignores the fact that $(dz)^2 = O(dt)$. To see what this means consider a Taylor expansion of dy at (x,t), i.e., totally differentiate the Taylor expansion of $f(x,t)$:

$$dy = f_x dx + f_t dt + \tfrac{1}{2} f_{xx}(dx)^2 + f_{xt} dx dt + \tfrac{1}{2} f_{tt}(dt)^2 + \text{ higher order terms.}$$

Terms of higher order than dt and dx are then ignored in the differential. In this case, however, the term $(dx)^2$ represents the square of a random variable that has expectation $\sigma^2 dt$ and, therefore, cannot be ignored. Including this term results in the differential

$$\begin{aligned} dy &= f_x dx + [\tfrac{1}{2} f_{xx}\sigma^2(x,t) + f_t]dt \\ &= [f_x \mu(x,t) + f_t + \tfrac{1}{2} f_{xx}\sigma^2(x,t)]dt + f_x \sigma(x,t) dz, \end{aligned}$$

a result known as Ito's Lemma. An immediate consequence of Ito's Lemma is that functions of Ito processes are also Ito processes (provided the functions have the appropriately continuous derivatives).

Multivariate versions of Ito's Lemma are easily defined. Suppose x is an n-vector valued process and z is a k-vector Wiener process (composed of k independent standard Wiener processes). Then μ is an n-vector valued function ($\mu : \Re^{n+1} \to \Re^n$) and σ is an $n \times k$ matrix valued function ($\sigma : \Re^{n+1} \to \Re^{n \times k}$). The instantaneous covariance of x is $\sigma\sigma^T$, which may be less than full rank.

It should be noted that some people define multivariate correlated Wiener processes and define σ to be a diagonal scaling matrix. Although mathematically equivalent, this approach is avoided here as it obscures the number of independent driving forces that define the vector x.

It will subsequently be shown that

$$x_{t+\Delta t} = x_t \exp(\mu \Delta t + \sigma \sqrt{\Delta t} v),$$

where $v \sim N(0,1)$.

For vector-valued x Ito's Lemma is

$$dy = \left[f_x\mu(x,t) + f_t + \tfrac{1}{2}\text{trace}\left(\sigma^T(x,t)f_{xx}\sigma(x,t)\right)\right]dt + f_x\sigma(x,t)dz,$$

(the only difference being in the second order term; derivatives are defined such that f_x is a $(1 \times n)$-vector). The lemma extends in an obvious way if y is vector valued.

Example: Computing Moments over Discrete Time Intervals
Ito's Lemma can be used to generate some simple results concerning Ito processes. For example, consider the case of geometric Brownian motion, defined as

$$dx = \mu x dt + \sigma x dz.$$

Define $y = \ln(x)$, implying that $\partial y/\partial t = 0$, $\partial y/\partial x = 1/x$ and $\partial^2 y/\partial x^2 = -1/x^2$. Applying Ito's Lemma yields the result that

$$dy = [\mu - \sigma^2/2]dt + \sigma dz.$$

This is a process with independent increments that are $N((\mu - \sigma^2/2)\Delta t, \sigma^2\Delta t)$. Hence a geometric Brownian motion process has conditional probability distributions that are lognormally distributed:

$$\ln(x_{t+\Delta t}) - \ln(x_t) \sim N\left((\mu - \sigma^2/2)\Delta t, \sigma^2\Delta t\right).$$

It is useful to have such an explicit expressions for the probability distribution of the discrete time increments, especially if one desires to estimate the parameters of the process (e.g., μ and σ). Unfortunately, it is rarely possible to derive such explicit expressions. In some cases, however, one can derive explicit expressions for the moments of the distribution. Consider the process defined by

$$dx = \rho(\mu - x)dt + \sigma(x,t)dz. \tag{10.3}$$

Taking expectations on both sides, allowing the expectation operator to pass through the linear differential operator and noting that $E\sigma(x,t)dz = 0$ yields

$$Edx = dEx = E\rho(\mu - x)dt = \rho(\mu - Ex)dt.$$

Letting $y = E_\tau x_t$, this expression can be written as the ordinary differential equation

$$dy/dt = \rho(\mu - y).$$

Together with the boundary condition that $y_\tau = x_\tau$, the solution is easily verified to be

$$y_t = \mu + e^{-\rho(t-\tau)}(y_\tau - \mu).$$

Hence the

$$E_\tau x_t = \mu + e^{-\rho(t-\tau)}(x_\tau - \mu).$$

Thus, an Ito process with an affine mean term is the continuous time analog to a first order difference process in discrete time. For $\rho = 0$, it is clear that x is a martingale process ($E_\tau x_t = x_\tau$), the equivalent of a simple unit root process in discrete time. For $\rho > 0$ the process is mean reverting, with a long run tendency to return to the value μ. In the limit as $\rho \to \infty$ the process fluctuates randomly around μ.

Second moments can be found by combining this approach with Ito's Lemma. Consider the special case of (10.3) with a constant variance term

$$dx = \rho(\mu - x)dt + \sigma dz.$$

To determine the variance of this process note that

$$Var_\tau(x_t) = E_\tau x^2 - (E_\tau x_t)^2$$

. Using Ito's Lemma

$$dx^2 = [2\rho(\mu - x)x + \sigma^2]dt + 2x\sigma dz.$$

The time derivative of the expectation of x^2 is therefore

$$\frac{dEx^2}{dt} = \sigma^2 - 2\rho Ex^2 + 2\rho\mu Ex.$$

The time derivative of the square of Ex is

$$\frac{d(Ex)^2}{dt} = -2\rho e^{-\rho(t-\tau)}(x_\tau - \mu)Ex.$$

Combining these expressions yields

$$\begin{aligned}\frac{dVar_\tau(x_t)}{dt} &= \sigma^2 - 2\rho Ex^2 + 2\rho\mu Ex + 2\rho e^{-\rho(t-\tau)}(x_\tau - \mu)Ex \\ &= \sigma^2 - 2\rho(Ex^2 - (Ex)^2)) \\ &= \sigma^2 - 2\rho Var_\tau(x_t).\end{aligned}$$

This is an ordinary differential equation with the boundary condition that $Var_\tau(x_\tau) = 0$, which is solved by

$$Var_\tau(x_t) = \frac{1 - e^{-2\rho(t-\tau)}}{2\rho}\sigma^2.$$

In the limit as $\rho \to 0$ this expression yields the familiar result that the conditional variance grows linearly in time:

$$\lim_{\rho \to 0} \frac{1 - e^{-2\rho(t-\tau)}}{2\rho} = t - \tau.$$

On the other hand, the limit as $\rho \to \infty$ demonstrates that the process becomes degenerate for large ρ, with the probability distribution being concentrated at the point μ. The long-run distribution of the process is found by letting $t \to \infty$), thereby demonstrating that the process has a long-run mean and variance (μ,$\sigma^2/2\rho$).

10.1.2 The Feynman-Kac Equation

Control theory in continuous time is typically concerned with problems which attempt to choose a control that maximizes a discounted return stream over time. It will prove useful, therefore, to have an idea of how to evaluate such a return stream for an arbitrary control. Consider the value

$$V(S_t, t) = E_t \left[\int_t^T e^{-\rho(\tau - t)} f(S_\tau) d\tau + e^{-\rho(T-t)} R(S_T) \right],$$

where

$$dS = \mu(S)dt + \sigma(S)dz.$$

An important theorem, generally known in economics as the Feynman-Kac Equation, but also known as Dynkin's Formula, states that $V(S)$ is the solution to the following partial differential equation[4]

$$\rho V(S,t) = f(S) + V_t(S,t) + \mu(S)V_S(S,t) + \tfrac{1}{2}\sigma^2(S)V_{SS}(S,t),$$

[4]The partial differential equation of this theorem has a linear parabolic form. Parabolic PDEs are ones that can be expressed in terms of the first time derivative and the second (and possibly lower) space derivatives. The term comes from the equation for a parabola $y = a + bx + cx^2$, substituting dt for y and dx for x. Other common forms of second order PDEs are hyperbolic and elliptic, both of which involve second order derivatives in both space and time.

with $V(S,T) = R(S)$. The function R here represents a terminal value of the state, i.e., a salvage value.[5]

By applying Ito's Lemma, the Feynman-Kac Equation can be expressed as:

$$\rho V(S,t) = f(S) + E[dV]/dt. \tag{10.4}$$

(10.4) has a natural economic interpretation. Notice that V can be thought of as the value of an asset that generates a stream of payments $f(S)$. The rate of return on the asset, ρV, is composed of two parts, $f(S)$, the current income flow and $E[dV]/dt$, the expected rate of appreciation of the asset. Alternative names for the components are the dividend flow rate and the expected rate of capital gains.

A version of the theorem applicable to infinite horizon problems states that

$$V(S_t) = E_t\left[\int_t^\infty e^{-\rho\tau} f(S) d\tau\right],$$

is the solution to the differential equation

$$\rho V(S) = f(S) + \mu(S)V_S(S) + \tfrac{1}{2}\sigma^2(S)V_{SS}(S).$$

Although more general versions of the theorem exist (for example see Duffie for a version with a state dependent discount rate), these will suffice for our purposes.

As with any differential equation, boundary conditions are needed to completely specify the solution. In this case, we require that the solution to the differential equation be consistent with the present value representation as S approaches its boundaries (often 0 and ∞ in economic problems). Generally economic intuition about the nature of the problem is used to determine the boundary conditions; we will discuss this issue more presently.

Example: Geometric Brownian Motion

Geometric Brownian motion is a particularly convenient stochastic process because it is relatively easy to compute expected values of reward streams. If S is governed by

$$dS = \mu S dt + \sigma S dz,$$

[5]The terminal time T need not be fixed, but could be a state dependent. Such an interpretation will be used in the discussion of optimal stopping problems (Section 11.2.3).

the expected present value of a reward stream $f(S)$ is the solution to

$$\rho V = f(S) + \mu S V_S + \tfrac{1}{2}\sigma^2 S^2 V_{SS}.$$

As this is a linear second order differential equation, the solution can be written as the sum of the solution to the homogeneous problem ($f(S) = 0$) and any particular solution that solves the non-homogeneous problem. The homogeneous problem is solved by

$$V(S) = A_1 S^{\beta_1} + A_2 S^{\beta_2},$$

where the β_i are the roots of the quadratic equation

$$\tfrac{1}{2}\sigma^2 \beta(\beta - 1) + \mu\beta - \rho = 0$$

and the A_i are constants to be determined by boundary conditions. For positive ρ, one of these roots is greater than one, the other is negative: $\beta_1 > 1$, $\beta_2 < 0$.

Consider the problem of finding discounted expected values of powers of S, i.e., where $f(S) = S^\gamma$ (assuming, momentarily, that it exists). It is easily verified that a particular solution is

$$V(S) = S^\gamma / (\rho - \mu\gamma - \tfrac{1}{2}\sigma^2\gamma(\gamma - 1)). \tag{10.5}$$

All that remains, therefore, is to determine the value of the arbitrary constants A_1 and A_2 that ensure the solution indeed equals the expected value of the reward stream. This is a bit tricky because it need not be the case that the expectation exist (the integral may not converge as its upper limit of integration goes to ∞). It can be shown, however, that the present value is well defined for $\beta_2 < \gamma < \beta_1$, making the numerator in (10.5) positive. Furthermore, the boundary conditions require that $A_1 = A_2 = 0$. Thus the particular solution is convenient in that it has a nice economic interpretation as the present value of a stream of returns.

10.1.3 Arbitrage Based Asset Valuation

An important use of continuous time methods results from powerful arbitrage conditions that can be derived in a simple and elegant fashion. Originally developed by Fisher Black and Myron Scholes, as well as by Robert Merton, to solve option pricing problems, arbitrage arguments apply much more broadly. Any assets that are based on the same underlying risks have values

that are related to one another in very specific ways. Although this clearly applies to financial derivatives such as options, it also applies more generally.

Consider two assets which have values V and W, both of which depend on the same random process S. Suppose that S is an Ito process, with[6]

$$dS = \mu_S dt + \sigma_S dz.$$

Under suitable regularity conditions, this implies that V and W are also Ito processes, with

$$dV = \mu_V dt + \sigma_V dz$$

$$dW = \mu_W dt + \sigma_W dz.$$

Suppose further that the assets generate income streams (dividends), which are denoted by δ_V and δ_W.

One can create a portfolio consisting of one unit of V and n units of W, the value of which is described by

$$dV + ndW = [\mu_V + n\mu_W]dt + [\sigma_V + n\sigma_W]dz.$$

This portfolio can be made risk free by the appropriate choice of n, specifically by setting the dz term to 0:

$$n = -\sigma_V/\sigma_W.$$

Because it is risk-free the portfolio must earn the risk-free rate of return. Therefore the capital appreciation on the portfolio plus its income stream must equal the risk free rate times the investment cost:

$$\left[\mu_V - \frac{\sigma_V}{\sigma_W}\mu_W\right] dt + \left[\delta_V - \frac{\sigma_V}{\sigma_W}\delta_W\right] dt = r\left[V - \frac{\sigma_V}{\sigma_W}W\right] dt$$

Divide by $\sigma_V dt$ and rearrange to conclude that

$$\frac{\mu_V + \delta_V - rV}{\sigma_V} = \frac{\mu_W + \delta_W - rW}{\sigma_W} = \theta(S,t).$$

[6]The following notational conventions are used. μ, σ and δ represent drift, diffusion and payouts associated with random processes; subscripts on these variables identify the process. V and W represent asset values, which are functions of the underlying state variables and time; subscripts refer to partial derivatives.

In other words, there is a function, θ, which depends on S and t, that is common to all assets whose values depend on S. θ can be interpreted as the market price of the risk in S.

To avoid arbitrage opportunities, any asset with value V that depends on S must satisfy

$$\mu_V + \delta_V = rV + \theta\sigma_V$$

This is a fundamental arbitrage condition that is interpreted as saying that the total return on V, $\mu_V + \delta_V$, equals risk free return plus a risk adjustment, $rV + \theta\sigma_V$.

Ito's Lemma provides a way to evaluate the μ_V and σ_V terms. Specifically,

$$\mu_V = V_t + \mu_S V_S + \tfrac{1}{2}\sigma_S^2 V_{SS}$$

and

$$\sigma_V = \sigma_S V_S.$$

Combining with the arbitrage condition and rearranging yields

$$rV = \delta_V + V_t + (\mu_S - \theta\sigma_S)V_S + \tfrac{1}{2}\sigma_S^2 V_{SS}. \tag{10.6}$$

This is the fundamental differential equation that any asset derived from S must satisfy, in the sense that it must be satisfies by any frictionless economy in equilibrium.

It is worth exploring the market price of risk function, θ, more carefully. θ is the market price of risk in S and therefore does not depend on the specific terms of any asset derived from S. This arbitrage framework is consistent, but more general than, the any specific market equilibrium such as the Capital Asset Pricing Model (CAPM). In the CAPM all assets have expected excess return (over the risk free rate) that is proportional to the expected excess return on the so-called market portfolio. The factor of proportionality, called the beta, is equal to the covariance of the excess returns on the asset and the market portfolio divided by the covariance of the market excess return:

$$\beta_S = \sigma_{SM}/\sigma_M^2.$$

Thus in the CAPM the following relationship holds

$$\mu_S + \delta_S - rS = \beta_S(\mu_M - rM).$$

(this assumes that the market portfolio is payout protected so $\delta_M = 0$). If we define the market price of risk on the market portfolio to be $\theta_M = (\mu_M - rM)/\sigma_M$, then in the CAPM the market price of risk on S will equal

$$\theta_S = \rho_{SM}\theta_M,$$

where $\rho_{SM} \equiv \sigma_{SM}/\sigma_S\sigma_M$ is the correlation between S and M. Thus, in the CAPM, there is a single market price of risk, θ_M, and the market price of any specific risk is θ_M times the correlation between the specific risk and the market risk.

It is important to note that, in general, S may or may not be the price of a traded asset. If it is the price of a traded asset then the arbitrage condition applies to S itself, so

$$\mu_S - \theta\sigma_S = rS - \delta_S.$$

Furthermore, the value of any asset, V, which is derived from S, satisfies the partial differential equation

$$rV = \delta_V + V_t + (rS - \delta_S)V_S + \tfrac{1}{2}\sigma_S^2 V_{SS}.$$

On the other hand, if S is not the price of a traded asset, but there is a traded asset or portfolio, W, that depends only on S, then the market price of risk, θ, can be inferred from the behavior of W:

$$\theta(S,t) = \frac{\mu_W + \delta_W - rW}{\sigma_W},$$

where δ_W is the dividend flow acquired by holding W.

Example: Black-Scholes Formula
Consider a non-dividend paying (or payout protected) stock ($\delta = 0$), the price of which follows

$$dS = \mu S dt + \sigma S dz,$$

where μ and σ are constants, so S follows a geometric Brownian motion (sometimes denoted $dS/S = \mu dt + \sigma dz$). The log differences, $\ln(S(t+\Delta t)) - \ln(S(t))$, are normally distributed with mean $(\mu - \frac{1}{2}\sigma^2)\Delta t$ and variance $\sigma^2\Delta t$.

A derivative asset is defined such that its value, $V(S,t)$, is a function of the state variable S implying that

$$rV = V_t + rSV_S + \tfrac{1}{2}\sigma^2 S^2 V_{SS}.$$

Suppose that the boundary condition is that $V(S,T) = \max(0, S-K)$, for some constant K and time T. This is the boundary condition for a European call option on S with a strike price of K. A call option has a payout at time T of $S-K$ if $S > K$ and 0 otherwise. It can be shown that

$$V(S,t) = S\Phi\left(d\right) - e^{-r\tau}K\Phi\left(d - \sigma\sqrt{\tau}\right)$$

where $\tau = T - t$,

$$d = \frac{\ln(S/K) + r}{\sigma\sqrt{\tau}} + \tfrac{1}{2}\sigma\sqrt{\tau},$$

and Φ is the standard normal CDF:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-\frac{1}{2}z^2}dz.$$

Some tedious algebra will demonstrate that

$$V_S = \Phi\left(d\right),$$

$$V_{SS} = \frac{\phi\left(d\right)}{\sigma S\sqrt{\tau}}$$

and

$$V_t = -\frac{\sigma S\phi\left(d\right)}{2\sqrt{\tau}} - re^{-r\tau}K\phi\left(d - \frac{\sigma\sqrt{\tau}}{2}\right),$$

where

$$\phi(x) = \Phi'(x) = \frac{e^{-\frac{1}{2}x^2}}{\sqrt{2\pi}}$$

(in the industry these are known as the delta, gamma and theta of the call option and are used in hedging portfolios of stocks). Using these expressions it is straightforward to verify that the partial differential equation above, including the boundary condition, is satisfied.

10.2 Probability Distributions for Ito Processes

10.2.1 Transition Distributions

Obtaining a policy function (optimal control feedback function) is only one of the goals of dynamic analysis. a topic taken up in the next chapter. Another important goal is to characterize the probability distributions of the state and control variables. For this we will need to use some results from the probability theory of stochastic processes. In particular, we will use the Kolmogorov forward equation, which describes the evolution of the probability distribution as it moves forward in time from some initial condition. This gives the transition density for any time horizon. By letting the time horizon go to infinity (assuming the process is stationary) we will obtain the long-run transition density.

Consider an arbitrary Ito process

$$dS = \mu(S,t)dt + \sigma(S,t)dz.$$

We are interested in knowing

$$Prob[S_T \leq b|S_t] = \int_{-\infty}^{b} f(S_T, T; S_t, t)dS.$$

Thus f represents the time t probability density function associated with S at time T, given the time t value of S is S_t.

The Kolmogorov forward equation is a partial differential equation that the transition probability density function must satisfy:

$$\begin{aligned} 0 &= \frac{\partial f(S,T)}{\partial T} + \frac{\partial \mu(S,T)f(S,T)}{\partial S} - \tfrac{1}{2}\frac{\partial^2 \sigma^2(S,T)f(S,T)}{\partial S^2} \\ &= f_T + (\mu_S - \sigma_S^2 - \sigma\sigma_{SS})f + (\mu - \sigma\sigma_S)f_S - \tfrac{1}{2}\sigma^2 f_{SS}. \end{aligned}$$

From the definition of the transition density function, f must have a degenerate distribution at $T = t$, i.e.,

$$f(S_t, t; S_t, t) = \delta(S_t),$$

where $\delta()$ is the Dirac function which concentrates all probability mass on a single point. It is often the case, however, that one wants only the long-run stationary distribution (assuming it exists), in which case the f_T term equals zero and the Kolmogorov equation reduces to a differential equation in S.

To illustrate the use of the Kolmogorov equation, consider processes that have Gaussian (normal) transition densities with mean $M(T)$ and variance $V^2(T)$. The Gaussian density is

$$f(S,T) = \frac{1}{\sqrt{2\pi}V(T)} \exp\left(-\frac{(S-M(T))^2}{2V^2(T)}\right).$$

with associated the partial derivatives:

$$\begin{aligned} f_S &= -\frac{S-M}{V^2} f \\ f_{SS} &= \frac{1}{V^2}\left(\frac{(S-M)^2}{V^2} - 1\right) f \\ f_T &= \left(\left(\frac{(S-M)^2}{V^2} - 1\right)\frac{V'}{V} + \frac{S-M}{V^2}M'\right) f. \end{aligned}$$

Substitute these definitions into the Kolmogorov equation and rearrange terms:

$$\left(\frac{(S-M)^2}{V^2} - 1\right)\left(\frac{V'}{V} - \frac{\sigma(S,T)^2}{2S^2}\right) - \frac{S-M}{V^2}\left(M' - \mu(S,T)\right) + \mu_S(S,T) = 0. \quad (10.7)$$

(10.7) can be satisfied when the drift term is linear in S and the variance term is constant in S:

$$\mu(S,t) = \mu_0(t) + \mu_1(t)S$$

$$\sigma(S,t) = \sigma(t).$$

Substituting these expressions into (10.7) yields

$$\left(\frac{(S-M)^2}{V^2} - 1\right)\left(\frac{V'}{V} - \frac{\sigma^2(T)}{2V^2}\right) - \left(\frac{(S-M)\left(S - \frac{M'-\mu_0(T)}{\mu_1(T)}\right)}{V^2} - 1\right)\mu_1(T) = 0.$$

If M satisfies $M = (M' - \mu_0(T))/\mu_1(T)$, or equivalently, $M' = \mu_1(T)M + \mu_0(T)$, this simplifies to

$$\left(\frac{(S-M)^2}{V^2} - 1\right)\left(\frac{V'}{V} - \frac{\sigma^2(T)}{2V^2} - \mu_1(T)\right) = 0.$$

Notice that the second term in () does not involve S; setting it equal to zero and rearranging yields

$$2VV' \equiv \left(V^2\right)' = \sigma^2(T) + 2\mu_1(T)V^2.$$

The Kolmogorov equation will therefore be satisfied if $M(t)$ and $V^2(t)$ are selected to satisfy the linear differential equations

$$M' = \mu_1(T)M + \mu_0(T)$$

$$\left(V^2\right)' = 2\mu_1(T)V^2 + \sigma^2(T),$$

with boundary conditions that $M(t) = S_t$ and $V^2(t) = 0$. It can be readily verified (see Appendix ??) that the solutions to these differential equations are

$$M(T) = \frac{\int_t^T \eta(\tau)\mu_0(\tau)d\tau + \eta(t)S_t}{\eta(T)}$$

$$V^2(T) = \frac{\int_t^T \eta^2(\tau)\sigma^2(\tau)d\tau}{\eta^2(T)},$$

where

$$\eta(T) = \exp\left(-\int^T \mu_1(\tau)d\tau\right).$$

10.2.2 Long-Run (Steady-State) Distributions

It is often not possible to find a closed form solution for the transition density. When the drift and diffusion terms are not functions of t, however, there may be a long-run density that represents the limit of the transition density as $T \to \infty$. We say "may be" because many Ito processes, including absolute and geometric Brownian motion, do not have long-run densities. For example, the mean, variance or other moments of the transition densities may become infinite in the long-run and hence no stationary density exists. Such is the case for absolute and geometric Brownian motion. On the other hand, some processes reach an absorbing barrier with positive probability and will have either discrete probability distributions (if they are absorbed with probability one) or a mixture of discrete and continuous distributions.

If a well defined long-run density exists, it will depend on S alone (not on T) and Kolmogorov forward equation becomes

$$\frac{d^2\sigma^2(S)\pi(S)}{dS^2} = 2\frac{d\mu(S)\pi(S)}{dS}$$

where π is the long-run density function. Integrating both sides and rearranging terms yields

$$\frac{d\sigma^2(S)\pi(S)}{\sigma^2(S)\pi(S)} = 2\frac{\mu(S)}{\sigma^2(S)}dS.$$

Integrating both sides again, taking the exponential of both sides and rearranging terms yields

$$\pi(S) = \frac{c}{\sigma^2(S)}\exp\left(2\int^S \frac{\mu(s)}{\sigma^2(s)}ds\right), \tag{10.8}$$

where c is chosen to ensure that π integrates to 1.[7]

To illustrate, consider again the process in which $\mu(S) = \alpha(m - S)$ and $\sigma(S) = \sigma$. The long run distribution is then equal to

$$\begin{aligned}
\pi(S) &= c\exp\left(2\int^S \frac{\alpha(m-s)}{\sigma^2}ds\right) \\
&= c\exp\left(\frac{2\alpha mS - \alpha S^2}{\sigma^2}\right) \\
&= \tilde{c}\exp\left(-\frac{2\alpha}{2\sigma^2}(S-m)^2\right),
\end{aligned}$$

which is recognizable as the normal distribution with mean m and variance $\sigma^2/2\alpha$.

Suppose instead that we are interested in an Ito process for which the log of S has a $N(m, v^2/2\alpha)$ long-run distribution. The density would then have the form

$$\pi(S) \propto \frac{1}{vS}\exp\left(-\frac{\alpha}{v^2}\left(\ln^2(S) - 2m\ln(S)\right)\right)$$

[7]See Merton (1975), Appendix B for further discussion. Merton discusses further regularity conditions on μ and σ (e.g., they are continuous and $\mu(0) = \sigma(0) = 0$). He points out that there is another solution to the Kolmogorov equation but that it must be zero when the probability of the boundaries of the state space is zero. This discussion is also related to the Feller classification of boundary conditions in the presence of singularities (see Bharucha-Reid, sec. 3.3 and Karlin and Taylor, chap.14).

To match terms within the exponential we must have

$$2\int^S \frac{\mu(S)}{\sigma^2(S)}ds = -\frac{\alpha}{v^2}\left(\ln(S)^2 - 2m\ln(S)\right).$$

Differentiating both sides, this implies that

$$2\frac{\mu(S)}{\sigma^2(S)} = 2\frac{\alpha}{v^2}\frac{m-\ln(S)}{S}. \tag{10.9}$$

Furthermore $\sigma^2(S)$ must be proportional to S to satisfy the term outside the exponential. Setting

$$\mu(S) = \alpha(m - \ln(S))$$

and

$$\sigma^2(S) = v^2 S$$

satisfies (10.9). Thus, the Ito process

$$dS = \alpha\left(m - \ln(S)\right)dt + v\sqrt{S}dz$$

has a log-normal long-run distribution with $\ln(S) \sim N(m, v^2/2\alpha)$.[8]

The expression for the mode of the long-run distribution is easily found from Equation (10.8). The mode, which maximizes the expression in (10.8) can equivalently be found by maximizing

$$\int^S \frac{\mu(s)}{\sigma^2(s)}ds - \ln\left(\sigma(S)\right).$$

[8]It should be noted, however, that

$$dx = \alpha(m-x)dt + vdz$$

and the log of S, where

$$dS = \alpha(m - \ln(S))dt + v\sqrt{S}dz$$

do not have the same transition densities, even though they have the same long-run density. To see this apply Ito's Lemma to the transformation $y = \ln(S)$ to obtain

$$dy = [\alpha(m-y) - \tfrac{1}{2}v^2]e^{-y}dt + ve^{-\frac{1}{2}y}dz,$$

the transition density of which is not known.

Differentiating both sides and setting the result equal to zero yields the condition

$$\mu(S) - \tfrac{1}{2}\frac{d\sigma^2(S)}{dS} = 0.$$

The value of S that solves this can be found either analytically or numerically.

In economics, state variables are often constrained to be positive and to exhibit mean reversion. A useful class of processes with these characteristics has variance term $\sigma^2 S^2$ and a drift term of the form

$$\alpha S f(S/m),$$

where f is a decreasing function with $f(1) = 0$ and with

$$\lim_{z \to 0} z f(z) = 0.$$

For such processes $S = 0$ is an absorbing barrier; once achieved the process remains at zero. An absorbing barrier at zero is a feature of many economically interesting processes, including stock values, where a zero value can be thought of as bankruptcy, and stocks of renewable resources, where a zero value indicates extinction. Examples of the graph of the instantaneous mean for such processes of the form

$$dS = \frac{\alpha}{\beta}\left(1 - \left(\frac{S}{m}\right)^{\beta}\right) S\,dt + \sigma S\,dz$$

were displayed in Figure 11.1 (page 328) for $\beta = -1, -0.5, ..., 2$. Figure 10.1 displays the corresponding long run probability distributions. Notice that, although $S = 0$ is an absorbing barrier, the probability of achieving the barrier is 0. Figure 10.2 displays the mean and mode for alternative values of β.

Table 10.1 displays long-run density functions, expected values and modes for several families of non-negative, mean reverting processes.

Example: Long-Run Sustainable Harvest

It is sometimes of interest to consider the long run consequences of control policies using the stationary distribution. Consider a renewable resource such as a fishery, the stock of which evolves according to

$$dS = \left[\frac{\alpha}{\beta}(1 - (S/m)^{\beta})S - q(S)\right] dt + \sigma S dz,$$

Table 10.1: Long-Run Densities For Selected Ito Processes

$\mu(S)$	$\sigma(S)$	$\pi(S)$	$E[S_\infty]$	$\max_S \pi(S)$
$\alpha(m - \ln(S))$	$\sigma\sqrt{S}$	$LN\,(m, \nu)$	$\exp(m + \nu/2)$	$\exp(m - 2\nu)$
$\alpha \ln(\frac{m}{S})S$	σS	$LN\,(\ln(m) - \nu, \nu)$	$\exp(-\nu/2)m$	$\exp(-2\nu)m$
$\frac{\alpha}{\beta}\left(1 - \left(\frac{S}{m}\right)^\beta\right) S$	σS	$GG\left(\frac{1}{\beta^2\nu} - \frac{1}{\beta}, (\beta^2\nu)^{\frac{1}{\beta}} m, \beta\right)$	$\frac{(\beta^2\nu)^{\frac{1}{\beta}}\Gamma\left(\frac{1}{\beta^2\nu}\right)}{\Gamma\left(\frac{1}{\beta^2\nu} - \frac{1}{\beta}\right)} m$	$(1 - 2\beta\nu)^{\frac{1}{\beta}} m$
$\frac{\alpha}{\beta}\left(\frac{1-\left(\frac{S}{m}\right)^\beta}{\gamma+\left(\frac{S}{m}\right)^\beta}\right) S$	σS	$GB2\left(\frac{1}{\gamma\beta^2\nu} - \frac{1}{\beta}, \gamma^{\frac{1}{\beta}} m, \beta, \frac{1}{\beta^2\nu} + \frac{1}{\beta}\right)$	$\frac{\gamma^{\frac{1}{\beta}}\mathrm{B}\left(\frac{1}{\gamma\beta^2\nu}, \frac{1}{\beta^2\nu}\right)}{\mathrm{B}\left(\frac{1}{\gamma\beta^2\nu} - \frac{1}{\beta}, \frac{1}{\beta^2\nu} + \frac{1}{\beta}\right)} m$	$\left(\frac{1-2\gamma\beta\nu}{1+2\beta\nu}\right)^{\frac{1}{\beta}} m$

Notes:
$\nu = \sigma^2/2\alpha$
$N(\mu, \sigma^2)$ denotes the normal (Gaussian) distribution
$LN(\mu, \sigma^2)$ denotes the log-normal distribution, i.e., $\ln(S)$ is $N(\mu, \sigma^2)$
Γ is the Gamma function with $\Gamma(a+1) = a\Gamma(a)$
B is the Beta function: $\mathrm{B}(a, d) = \Gamma(a)\Gamma(d)/\Gamma(a+d)$
Generalized Gamma: $GG(S; a, b, c) = \frac{c(S/b)^{ac-1}\exp(-(S/b)^c)}{b\Gamma(a)}$
Generalized Beta-2: $GB2(S; a, b, c, d) = \frac{c(S/b)^{ac-1}(1+(S/b)^c)^{-(a+d)}}{b\mathrm{B}(a,d)}$
The Generalized Gamma distribution requires that $\nu < 1$; otherwise the process goes to zero with probability one. With $\beta = 1$, $E[S_\infty] = (1 - \nu)m$ (the 2-parameter Gamma distribution).
The Generalized Beta-2 distribution requires that $\gamma\nu < 1$. With $\beta = 1$, $E[S_\infty] = (1 - \gamma\nu)m$ (the Generalized Pareto distribution).

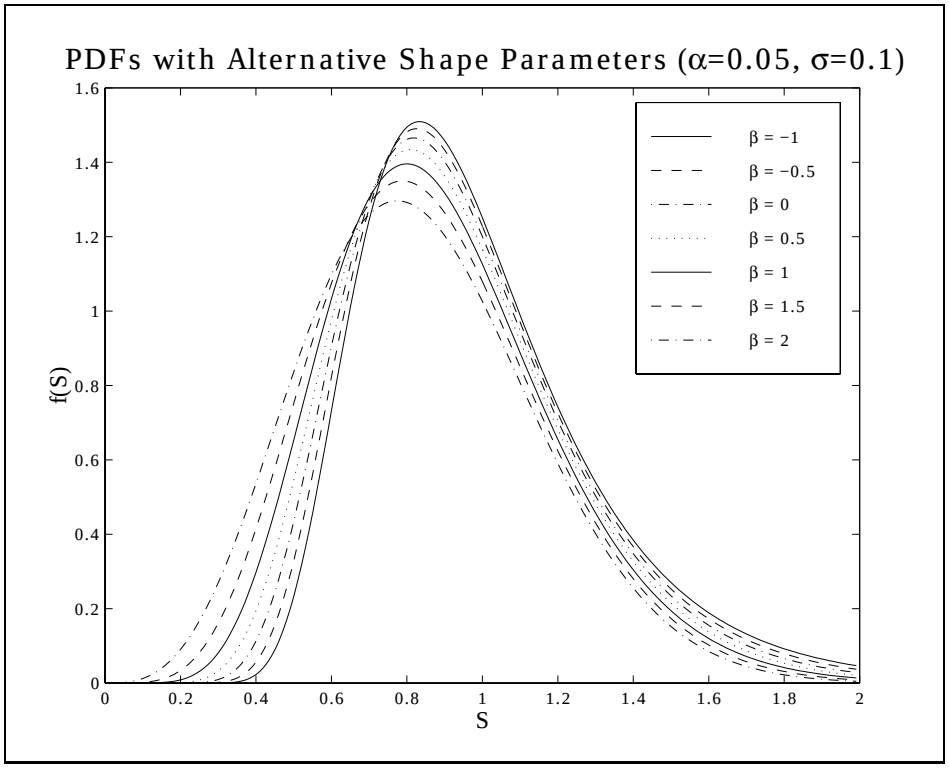

Figure 10.1

where $q(S)$ is the harvest rate. The biological growth process (with $q(S) = 0$) is the mean reverting process discussed in the example on page 324.

For regulatory purposes it is of interest to know what harvest rate maximizes the long-run average harvest level. This would require maximization over the function $q(S)$; a simpler problem is to find the maximizing constant proportional harvest rate ($C(S) = cS$), which requires maximization over the constant c. This is facilitated by noting that, for this harvest function, the long run average harvest is c times the long-run average stock level.

The mean term for the process can be put in the form

$$\tilde{\mu}(S) = \frac{\tilde{\alpha}}{\beta}\left(1 - \left(\frac{S}{\tilde{m}}\right)^{\beta}\right) S$$

by setting $\tilde{\alpha} = (\alpha - \beta c)$ and $\tilde{m} = (1 - \beta c/\alpha)^{1/\beta} m$. Using Table 10.1, the

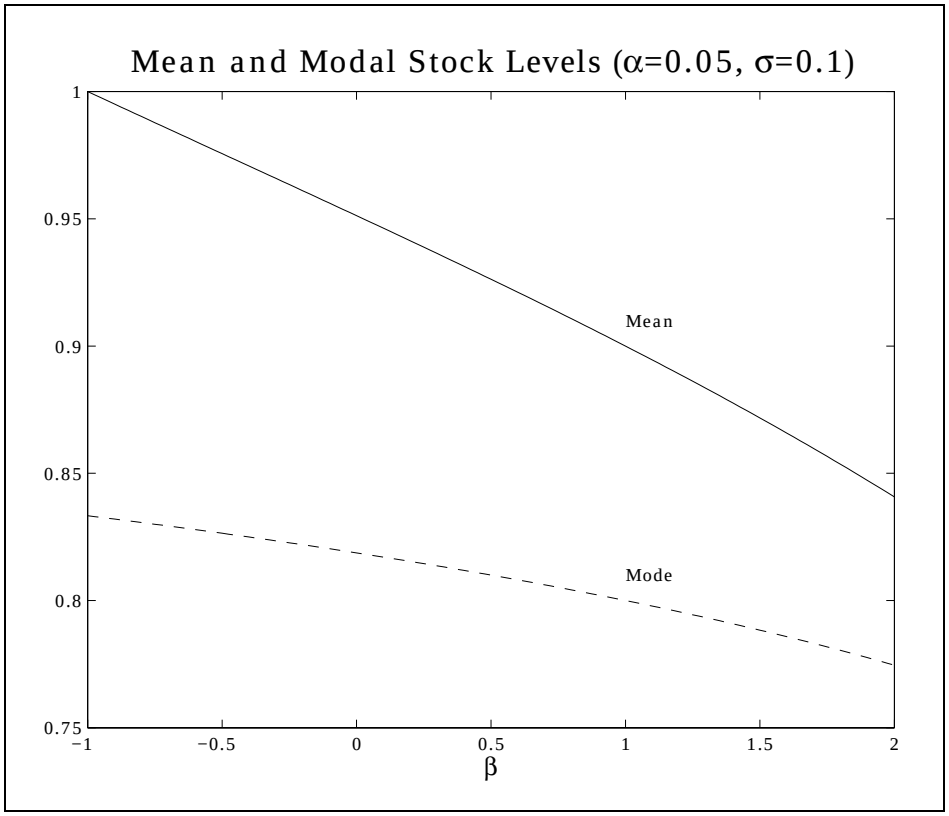

Figure 10.2

long-run average harvest is

$$\frac{\left(\frac{\beta^2\sigma^2}{\alpha}\right)^{\frac{1}{\beta}}\Gamma\left(\frac{2(\alpha-\beta c)}{\beta^2\sigma^2}\right)}{\Gamma\left(\frac{2(\alpha-\beta c)}{\beta^2\sigma^2}-\frac{1}{\beta}\right)}cm.$$

Maximizing this with respect to c is equivalent to the following problem

$$\max_c \ln\Gamma\left(\frac{2(\alpha-\beta c)}{\beta^2\sigma^2}\right)-\ln\Gamma\left(\frac{2(\alpha-\beta c)}{\beta^2\sigma^2}-\frac{1}{\beta}\right)+\ln(c).$$

The FOC for this problem can be written[9]

$$-\frac{2}{\beta\sigma^2}\left\{\psi\left(\frac{2(\alpha-\beta c)}{\beta^2\sigma^2}\right)-\psi\left(\frac{2(\alpha-\beta c)}{\beta^2\sigma^2}-\frac{1}{\beta}\right)\right\}+\frac{1}{c}=0,$$

which can be solved for c using a standard root-finding algorithm (see Section ??). In the special case that $\beta = 1$ the solution has the particularly simple form[10]

$$c=\frac{\alpha}{2}-\frac{\sigma^2}{4}. \tag{10.10}$$

It is useful to note that for $\sigma^2 > 2\alpha$ the resource has a non-zero probability of extinction even if no extraction occurs. Thus, in cases in which the possibility of extinction occurs, the optimal sustainable (constant catch rate) policy is to not catch at all.[11]

Values of the maximum sustainable average harvest rate for alternative β and σ are shown in Figure 10.3. The fact that the catch rate increases as β decreases is explainable by recalling from Figure 11.1 that lower values of β result in a stock that recovers from low levels more quickly and dies off from high stock levels more slowly.

Code Box 10.1: Maximal Sustainable Harvest Rate

---

[9]The psi function $\psi(x)$ is the derivative of the log of the gamma function; see Abromowitz and Stegun for details. The function file PSI.M is provided to evaluate this function.

[10]It is helpful to note that $\psi(x) - \psi(x-1) = 1/(x-1)$.

[11]Solutions for other values of β are possible, though tedious. For example, it can be shown that for $\beta = \frac{1}{2}$

$$c=\frac{1}{12}\left(16\alpha-3\sigma^2-\sqrt{64\alpha^2-24\alpha\sigma^2+3\sigma^4}\right)$$

In the limiting case as $\beta \to 0$, the mean process approaches $\alpha S \ln(m/S) - cS$ and the mean harvest rate approaches

$$e^{\left(\frac{\sigma^2}{4\alpha}-\frac{c}{\alpha}\right)}cm$$

which is maximized at $c = \alpha$. It can also be shown that as $\sigma \to 0$, c approaches $\alpha/(1+\beta)$.

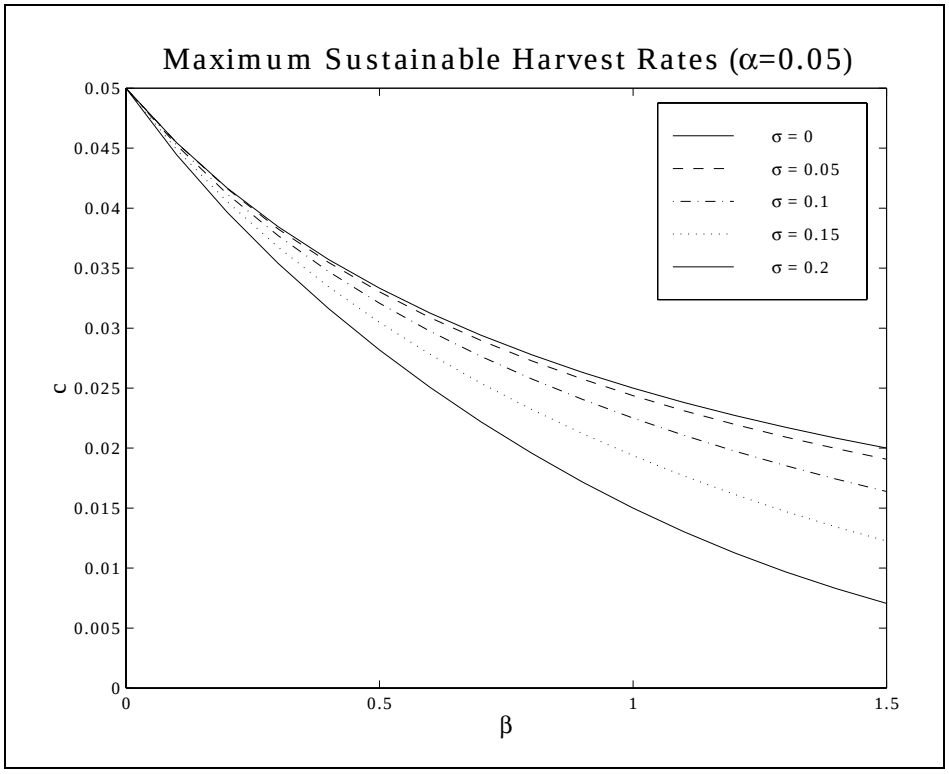
Maximum Sustainable Harvest Rates (α=0.05)
c
β
σ = 0
σ = 0.05
σ = 0.1
σ = 0.15
σ = 0.2
0.05
0.045
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
0
0.5
1
1.5

Figure 10.3

10.3 End Notes

10.3.1 Bibliographic Notes

Many books contain discussions of Ito stochastic calculus. A useful reference with finance related applications is Hull; this has a particularly good discussion of arbitrage conditions. At a more advanced level see Duffie; the discussion of the Feynman-Kac formula draws heavily on this source.

A brief but useful discussion of steady-state distributions is found in Appendix B of Merton (1985). For more detail, including discussion of boundary issues, see Karlin and Taylor, chapter 15. Early work in this area is contained in several papers by Feller.

[INCOMPLETE]

10.3.2 References

Cox, D.R. and H.D. Miller. The Theory of Stochastic Processes. John Wiley and Sons, New York. 1965.

Feller, William. "Two Singular Diffusion Problems." Annals of Mathematics. 54(1951): 173-182.

Merton, Robert C. "An Asymptotic Theory of Growth Under Uncertainty." Review of Economic Studies. 42(1975):375-393.

[INCOMPLETE]

Exercises

1. Consider the continuous time optimal control problem of determining the investment policy that maximizes the discounted stream of income:

$$\begin{aligned} \max_{x(t)} \quad & \int_0^{\infty} f(x(t), t) \\ s.t. \quad & \frac{dk(t)}{dt} = x(t) - \rho \cdot k(t) \\ & x(t) \geq 0 \\ & k(0) = k_0 \end{aligned}$$

   where, in period t, $k(t)$ is the level of capital stock, $x(t)$ is the optimal rate of investment, $f(k, t)$ is the income generated from a capital stock k, and ρ is the continuous rate of capital decay.

   Formulate the Hamiltonian associated with the optimal control problem. Give an explicit economic interpretion the costate variable and the Hamiltonian. Apply the K-K-T theorem to the maximization problem embedded in Pontryagin's Maximum Principle to show that for an optimal path, $x(t) \cdot \lambda(t) = 0$ for all t. In light of the assertion just proved, state and interpret the costate equation.

2. Consider an economy of price taking firms facing adjustment costs and a downward-sloping demand curve for their identical products. The competitive rational expectations equilibrium for this economy is characterized by the capital accumulation path that solves the following dynamic optimization problem:

$$\max_{I(\cdot)} \quad \int_0^{\infty} \exp(-rt)\{W(K) - pI - \frac{1}{2}aI^2\}dt$$

$$\begin{aligned} s.t. \quad & \dot{K} = I - \eta K \\ & K(0) = K_0 \end{aligned}$$

where

$$\begin{aligned} K(t) &= \text{capital stock at time t} \\ I(t) &= \text{capital investment rate at time t (may be negative)} \\ W(K) &= \int_0^{F(K)} D(Q)dQ \quad \text{is consumer surplus} \\ D(Q) &= \text{is the inverse demand function} \quad (D(Q) > 0, D'(Q) < 0) \\ F(K) &= \text{is the production function} \quad (F(0) = 0, F'(K) > 0) \\ a &> 0 \quad \text{is the cost of adjustment coefficient} \\ r &> 0 \quad \text{is the interest rate} \\ p &> 0 \quad \text{is the unit price of new capital} \\ \eta &> 0 \quad \text{is the depreciation rate of capital} \end{aligned}$$

Perform a comparative dynamics analysis of the steady-state equilibrium, using a phase diagram to illustrate your discussion. Linearize about the steady state, deriving and interpreting the eigenvalues and eigenvectors corresponding to the linear approximation. How does the steady-state level of capital stock change as a, r, p, or η increase? How does the speed of adjustment to steady-state change as a, r, p, or η increase?

3. Suppose that a variable is governed by

$$dS = \mu S dt + \sigma S dz.$$

a) Show that

$$E_0 [S_t] = S_0 e^{\mu t}.$$

b) Use (a) to show that

$$E_0 \left[\int_0^t e^{-\rho\tau} S_\tau d\tau \right] = (1 - e^{-\rho t}) \frac{S_0}{\rho - \mu}.$$

4. Pricing Bonds

   Define $P(t,T)$ to be the current (time t) price of a pure discount bond maturing at time T, i.e., a bond that pays \$1 at time T. The price of a bond of any maturity depends on the instantanous interest rate, r. It can be shown that

   $$P(r,t,T) = \hat{\mathrm{E}}\left[\exp\left(-\int_t^T r(\tau)d\tau\right)\right],$$

   where the expectation is taken with respect to the risk adjusted process governing the instanteous interest rate. Assuming that this process is

   $$dr = \mu(r,t)dt + \sigma(r,t)dz$$

   an extended version of the Feyman-Kac Formula implies that P is the solution to

   $$rP = P_t + \mu(r,t)P_r + \tfrac{1}{2}\sigma^2(r,t)P_{rr},$$

   subject to the boundary condition that $P(r,T,T) = 1$.

   Suppose that the instantaneous interest rate process is

   $$dr = \alpha(m-r)dt + \sigma dz.$$

   Show that P has the form

   $$P(r,t,T) = A(t,T)\exp(-B(t,T)r)$$

   and, in doing so, determine the functions A and B.

5. Given the setting of the previous problem, suppose we take the instantaneous interest rate process to be

   $$dr = \alpha(m-r)dt + \sigma\sqrt{r}dz.$$

   Verify numerically that P has the form

   $$P(r,t,T) = A(t,T)\exp(-B(t,T)r)$$

with

$$A(t,T) = \left(\frac{2\gamma e^{(\gamma+a)(T-t)/2}}{(\gamma+a)(e^{\gamma(T-t)}-1)+2\gamma} \right)^{2ab/\sigma^2}$$

and

$$B(t,T) = \frac{2(e^{\gamma(T-t)}-1)}{(\gamma+a)(e^{\gamma(T-t)}-1)+2\gamma},$$

where $\gamma = \sqrt{a^2+2\sigma^2}$.

6. A futures contract maturing in τ periods on a commodity whose price is governed by

$$dS = \mu(S,t)dt + \sigma(S,t)dz$$

can be shown to satisfy

$$V_\tau(S,\tau) = (rS - \delta(S,t))V_S(S,\tau) + \tfrac{1}{2}\sigma^2(S,t)V_{SS}(S,\tau)$$

subject to the boundary condition $V(S,0) = S$. Here δ is interpreted as the convenience yield, i.e., the flow of benefits that accrue to the holders of the commodity but not to the holders of a futures contract. Suppose that the volatility term is

$$\sigma(S,t) = \sigma S.$$

In a single factor model one assumes that δ is a function of S and t. Two common assumptions are

$$\delta(S,t) = \delta$$

and

$$\delta(S,t) = \delta S.$$

In both cases the resulting V is linear in S. Derive explicit expressions for V given these two assumptions.

7. Continuing with the previous question, suppose that the convenience yield is

$$\delta(S,t) = \delta S$$

where δ is a stochastic mean-reverting process governed by

$$d\delta = \alpha(m-\delta)dt + \sigma_\delta dw,$$

with $\mathrm{E}dzdw = \rho\sigma\sigma_\delta$. Furthermore, suppose that the market price of the convenience yield risk is a constant θ. Then the futures price solves

$$V_\tau = (r-\delta)SV_S + (\alpha(m-\delta) - \theta)\, V_\delta + \tfrac{1}{2}\sigma^2 S^2 V_{SS} + \rho\sigma\sigma_\delta S V_{S\delta} + \tfrac{1}{2}\sigma_\delta^2 V_{\delta\delta},$$

with $V(S,0)$=S.

Verify that the solution has the form $V = \exp(A(\tau) - B(\tau)\delta)S$ and in doing so derive expression for $A(\tau)$ and $B(\tau)$.

8. Suppose that

$$dS = \mu dt + \sigma dz,$$

where μ and σ are constants. Show that the transition probability distribution is normal and determine it mean and variance.

9. Suppose that

$$dS = \alpha(m-S)dt + \sigma dz$$

(in the notation of Section 10.2.1 $\mu_0 = \alpha m$ and $\mu_1 = -\alpha$). Show that the transition probability distribution is normal and determine it mean and variance. Take limits as $T \to \infty$ to determine the long-run distribution.

10. Suppose that

$$dS = \sigma e^{-\alpha t} dz.$$

Show that the transition distribution is normal and determine it mean and variance.

11. Verify the solutions for maximum average sustainable harvest rates for $\beta = 1$, $\frac{1}{2}$ and 0 given in Equation (10.10) and Footnote 11.

Chapter 11

Continuous Time Dynamic Models: Theory

11.1 Stochastic Control

On an intuitive level, continuous time optimization methods can be viewed as simple extensions of discrete time methods. In continuous time one replaces the summation over time in the objective function with an integral evaluated over time and the difference equation defining the state variable transition function with a differential equation. For non-stochastic models, the optimization problem is[1]

$$\max_{x(S,t)} \int_0^T e^{-\rho t} f(S,x)dt + e^{-\rho T} R(S(T)), \text{ s.t. } dS = g(S,x)dt,$$

where S is the state variable (the state), x the control variable (the control), f is the reward function, g the state transition function and R is a terminal period "salvage" value. The time horizon, T, may be infinite (in which case R has no meaning) or it may be state dependent and must be determined endogenously (see Section 11.2.3 on optimal stopping).

For non-stochastic problems, optimal control theory and its antecedent, the calculus of variations, have become standard tools in economists mathematical toolbox. Unfortunately, neither of these methods lends itself well to extensions involving uncertainty. The other alternative for solving such

[1]We cover here the more common discounted time autonomous problem. The more general case is developed as an exercise.

problems is to use continuous time dynamic programming. Uncertainty can be handled in an elegant way if one restricts oneself to modeling that uncertainty using Ito processes. This is not much of a restriction because the family of Ito processes is rather large and can be used to model a great variety of dynamic behavior (the main restriction is that it does not allow for jumps). Furthermore, we will show that for deterministic problems, optimal control theory and dynamic programming are two sides of the same coin and lead to equivalent solutions. Thus, the only change needed to make the problem stochastic is to define the state variable, S, to be a controllable Ito process, meaning that the control variable, x, influences the value of the state:[2]

$$dS = g(S, x)dt + \sigma(S)dz.$$

To develop the solution approach on an intuitive level, notice that for problems in discrete time, Bellman's equation can be written in the form

$$V(S,t) = \max_x \left(f(S,x)\Delta t + \frac{1}{1+\rho\Delta t} E_t[V(S_{t+\Delta t}, t+\Delta t)] \right).$$

Multiplying this by $(1+\rho\Delta t)/\Delta t$ and rearranging:

$$\rho V(S,t) = \max_x \left(f(S,x,t)(1+\rho\Delta t) + \frac{E_t[V(S_{t+\Delta t}, t+\Delta t) - V(S,t)]}{\Delta t} \right).$$

Taking the limits of this expression at $\Delta t \to 0$ yields the continuous time version of Bellman's equation:

$$\rho V(S,t) = \max_x \left(f(S,x,t) + \frac{E_t dV(S,t)}{dt} \right). \tag{11.1}$$

If we think of V as the value of an asset on a dynamic project, Bellman's equation states that the rate of return on V (ρV) must equal the current income flow to the project (f) plus the expected rate of capital gain on the asset ($E[dV]/dt$), both evaluated using the best management strategy (i.e., the optimal control). Thus, Bellman's equation is a kind of intertemporal arbitrage condition.[3]

[2] A more general form would allow x to influence the diffusion as well as the drift term; this can be handled in a straightforward fashion but makes exposition somewhat less clear.

[3] It is important to note that the arbitrage interpretation requires that the discount rate, ρ, be appropriately chosen (see Section 10.1.3 for further discussion).

By Ito's Lemma

$$dV = [V_t + g(S,x)V_S + \tfrac{1}{2}\sigma(S)^2 V_{SS}]dt + \sigma(S)V_S dz.$$

Taking expectations and dividing by dt we see that the term $E_t dV(S,t)/dt$ can be replaced, resulting in the following form for Bellman's equation in continuous time:[4]

$$\rho V = \max_x f(S,x) + V_t + g(S,x)V_S + \tfrac{1}{2}\sigma^2(S)V_{SS}. \tag{11.2}$$

The maximization problem is solved in the usual way by setting the first derivative equal to zero:

$$f_x(S,x) + g_x(S,x)V_S = 0. \tag{11.3}$$

Combining this with

$$\rho V = f(S,x) + V_t + g(S,x)V_S + \tfrac{1}{2}\sigma^2(S)V_{SS} \tag{11.4}$$

results in two functional equations that must be solved for to yield the two functions: the value function $V(S,t)$ and the optimal policy function $x^*(S,t)$.[5]

If a solution to the maximization problem can be found of the form

$$x = x(S, V_S)$$

it may be useful to form the *concentrated Bellman equation*:

$$\rho V = f(S, x(S,V_S)) + V_t + g(S, x(S,V_S))V_S + \tfrac{1}{2}\sigma^2(S)V_{SS}. \tag{11.5}$$

Notice that the concentrated Bellman equation is non-linear whereas the Bellman equation is linear in the value function and its partial derivatives. The usefulness of the concentrated Bellman Equation will depend on whether it is easier to solve a single nonlinear PDE or a linear PDE combined with a functional equation not involving derivatives.

Notice that Bellman's Equation is not stochastic; the expectation operator and the randomness in the problem have been eliminated by using Ito's Lemma. As with discrete time versions the state transition equation is incorporated in Bellman's equation. This effectively transforms a stochastic

[4]Also known as the Hamilton-Jacobi-Bellman equation.

[5]It may be puzzling why the max operator is dropped from 11.4 until it is noted that 11.3 must be satisfied simultaneously, i.e., the optimized value of $x(S,t)$ is used in 11.4.

dynamic problem into a deterministic one. If there are additional constraints on the state variables they typically can be handled in the usual way (using Lagrange multipliers and, for inequality constraints, Karush-Kuhn-Tucker type conditions). Constraints on the control on somewhat more problematic (they are discussed in the inventory example in Section 11.2.3).

In finite time horizon problems, the value function is a function of time. In infinite time horizon problems, however, the value function becomes time invariant, implying that V is a function of S alone and thus $V_t = 0$. Thus the Bellman's Equation simplifies to

$$\rho V = \max_x f(S,x) + g(S,x)V_S + \tfrac{1}{2}\sigma^2(S)V_{SS}.$$

11.1.1 Relation to Optimal Control Theory

It is worth spending some time relating the dynamic programming approach to optimal control theory. As stated previously, optimal control theory is not naturally applied to stochastic problems but it is used extensively in deterministic ones. The Bellman equation in the determinstic case is

$$\rho V = \max_x f(S,x) + V_t + g(S,x)V_S,$$

where x is evaluated at its optimal level. Suppose we totally differentiate the marginal value function with respect to time:

$$\frac{dV_S}{dt} = V_{St} + V_{SS}\frac{dS}{dt} = V_{St} + V_{SS}g(S,x).$$

Now apply the Envelope Theorem to the Bellman equation to determine that

$$\rho V_S = f_S(S,x) + V_{tS} + g(S,x)V_{SS} + V_S g_S(S,x).$$

Combining these expressions and rearranging yields

$$\frac{dV_S}{dt} = \rho V_S - f_S - V_S g_S. \tag{11.6}$$

This can be put in a more familiar form by defining $\lambda = V_S$. Then (11.6), combined with the FOC for the maximization problem and the state transition equation can be written as the following system

$$0 = f_x(S,x) + \lambda g_x(S,x)$$

$$\frac{d\lambda}{dt} = \rho\lambda - f_S(S,x) - \lambda g_S(S,x)$$

and

$$\frac{dS}{dt} = g(S,x).$$

These relationships are recognizable as the Hamiltonian conditions from optimal control theory, with λ the costate variable representing the shadow price of the state variable (expressed in current value terms).[6]

The message here is that dynamic programming and optimal control theory are just two approaches to arrive at the same solution. It is important to recognize the distinction between the two approaches, however. Optimal control theory leads to three equations, two of which are ordinary differential equations in time. Optimal control theory therefore leads to expressions for the time paths of the state, control and costate variables as functions of time: $S(t)$, $x(t)$ and $\lambda(t)$. Dynamic programming leads to expressions for the control and the value function (or its derivative, the costate variable) as functions of time and the state. Thus dynamic programming leads to decision rules rather than time paths. In the stochastic case, it is precisely the decision rules that are of interest, because the future time path, even when the optimal control is used, will always be uncertain. For deterministic problems, however, DP involves solving partial differential equations, which tend to present more challenges than ordinary differential equations.

11.1.2 Boundary Conditions

The Bellman's equation expresses the optimal control in terms of a differential equation. In general, there will be many solutions, many of which are useless to us. Furthermore, from a numerical point of view, without boundary conditions imposed on the problem, it will be luck as to whether the derived solution is indeed the correct one. Unfortunately, the literature on this topic is incomplete and boundary conditions are often justified by economic rather than mathematical reasoning. For example, consider a case in which one is extracting a resource with a stochastic price. Suppose also that the price has an absorbing barrier at $P = 0$ (e.g., $dP = \alpha(m - P)Pdt + \sigma Pdz$). The value of the inventory is a function of the level of the inventory and the price:

[6]See Kamien and Schwartz, pp. 151-152 for further discussion.

$V(I,P)$. The reward function is Pq, where $dI = -qdt$, so the control q is the rate of extraction. It is obvious that the stream of profits generated by selling from an inventory will be zero if the price is zero because, once zero is reached, the price is zero forever and the inventory is therefore worthless. Also, if the inventory reaches zero it is worthless. We see, therefore, that

$$V(I,0) = V(0,P) = 0.$$

We would still need to determine upper boundaries, which we discuss further in the example on page 324.

Many problems in economics specify a reward function that has a singularity at an endpoint. Typical examples include utility of consumption functions for which zero consumption is infinitely bad. The commonly used constant relative risk aversion family of utility functions

$$U(c) = (c^\lambda - 1)/\lambda$$

(with $\ln(c)$ when $\lambda = 0$) is a case in point. Again, economic reasoning would suggest that if consumption is derived from a capital or resource stock and that stock goes to zero, consumption must also go to zero and hence the value of a zero stock, which equals the discounted stream of utility from that stock must be $-\infty$. Furthermore, the marginal value of the stock when the stock gets low becomes quite large, with $V_S = \infty$ as $S \to 0$. Although this reasoning makes good sense from an economic perspective, it raises some difficulties for numerical analysis.

As a rule of thumb, one needs to impose a boundary condition for each derivative that appears in Bellman's equation. For a single state problem, this means that there are two boundary conditions needed. In a two-dimensional problem with only one stochastic state variable, we will need two boundary conditions for the stochastic state and one for the non-stochastic one. For example, suppose Bellman's equation has the form

$$\rho V = f(S,R,x) + g(S,R,x)V_R + \mu(S)V_S + \tfrac{1}{2}\sigma^2 S^2 V_{SS}.$$

To completely specify the problem we could impose a condition at a point $R = R_b$, e.g. $V(S,R_b) = H(S)$ and conditions at $S = \underline{S}$ and $S = \overline{S}$, say $V_{SS}(\underline{S}) = V_{SS}(\overline{S}) = 0$.

Like all rules of thumb, however, there are exceptions. The exceptions tend to arise in singular problems, when the variance term vanishes at a

boundary. For example, it may not be necessary to impose explicit boundary conditions when the state variable is governed by

$$dS = \mu(S, x)dt + \sigma S dz,$$

where $\mu(0, 0) = 0$ and x is constrained such that $x = 0$ if $S = 0$. Zero is a natural absorbing state for this process, meaning that if $S(t) = 0$ then $S(\tau) = 0$ for all $\tau > t$. In this case, it may not be necessary to impose conditions on the boundary at $S = 0$.

An intuitive way to think of this situation is that a second order differential equation becomes effectively first order as the variance goes to zero. We may, therefore, not need to impose further conditions to achieve a well defined solution. Feller devised a classification scheme for diffusion processes with singular boundaries (see discussion by Bharucha-Reid, sec. 3.3, and Karlin and Taylor, Chap 15.). Although the literature is incomplete on this issue, a rule of thumb is that, if a boundary is inaccessible, meaning that the probability is zero that the process (naturally) will achieve the boundary, no boundary condition need be imposed. Several examples we will discuss have singular boundary conditions.

It is perhaps clear that in continuous time stochastic problems, defining the appropriate differential equation is the easy part and specifying the correct boundary conditions is the tricky part.

11.1.3 Choice of the Discount Rate

The choice of the appropriate discount rate to use in dynamic choice problems has been a topic of considerable discussion in the corporate finance literature. The arbitrage theory discussed in the previous chapter has been fruitfully applied to this issue. In particular, there is an equivalence between the choice of a discount rate and the price of risk assigned to the various sources of risk affecting the problem.

In general, if there is a market for assets that depend on a specific risk, S, then arbitrage constrains the choice of the discount rate. If an inappropriate discount rate is used, a potential arbitrage opportunity is created by either overvaluing or undervaluing the risk of the project. To see this note that the concentrated Bellman's equation for a dynamic project can be written

$$\rho V = \delta_V + V_t + \mu_S V_S + \tfrac{1}{2}\sigma_S V_{SS},$$

where $\delta_V = f(S, x^*, t)$ and $\mu_x = g(S, x^*, t)$. To avoid arbitrage, however, (10.6) must hold. Together these relationships imply that

$$\rho = r + \theta\sigma_S V_S/V = r + \theta\sigma_V/V \tag{11.7}$$

In practice we can eliminate the need to determine the appropriate discount rate by using the risk-free rate as the discount rate and acting as if the process S has instantaneous mean of either

$$\hat{\mu}_S = \mu_S - \theta_S\sigma_S$$

or

$$\hat{\mu}_S = rS - \delta_S.$$

Which form is more useful depends on whether it is easier to obtain estimates of the market price of risk for S, θ_S, or income stream generated by S, δ_S. The latter, however, is only possible if S is itself the value of an asset, whereas the former can be estimated (in principle) if there is some traded asset the value of which depends on S.

Even if the project involves a non-traded risk, it may be easier to guess the market price of that risk than to define the appropriate discount rate. For example, if the risk is idiosyncratic and hence can be diversified away, then a well-diversified agent would set the market price of risk to zero. An appropriate discount rate is particularly difficult to select when there are multiple source of risk (state variables) because the discount rate becomes a complicated function of the various market prices of risk.

Having said that, there may be cases in which the appropriate discount rate is easier to set. For firm level capital budgeting, the discount rate is the required rate of return on the project and, in a well functioning capital market, should equal the firm's cost of capital. Thus the total return on the project must cover the cost of funds:

$$\rho V = \delta_V + \mu_V = rV + \theta_S\sigma_V.$$

The cost of funds, ρ, therefore implicitly determines the market price of risk (using 11.7).

Summarizing, there are three alternative cases to consider:

1. S is a traded asset for which

$$\mu_S - \theta\sigma_S = rS - \delta_S$$

2. S is not a traded asset but there is a traded asset the value of which, W, depends on S and the market price of risk can be determined according to

$$\theta = (\mu_W + \delta_W - rW)/\sigma_W$$

3. S represents a non-priced risk and either θ or ρ must be guessed

When S is a controllable Ito process, the payment stream, $\delta(S,t)$, becomes $f(S,x,t)$ and the drift term, $\mu(S,t)$, becomes $g(S,x,t)$. There are three forms of Bellman's equation:

A) $$rV = \max_x f(S,x,t) + V_t + V_S\,(rS - \delta_S) + \tfrac{1}{2}V_{SS}\sigma^2(S,t)$$

B) $$rV = \max_x f(S,x,t) + V_t + V_S\,(g(S,x,t) - \theta\sigma(S,t)) + \tfrac{1}{2}V_{SS}\sigma^2(S,t)$$

C) $$\rho V = \max_x f(S,x,t) + V_t + V_S g(S,x,t) + \tfrac{1}{2}V_{SS}\sigma^2(S,t)$$

Any of the three forms can be used when S is a traded asset, although (A) and (B) are preferred in that they rely on market information rather than on guesses concerning the appropriate discount rate. When S is not a traded asset but represents a risk priced in the market, (B) is the preferred form although (C) can be used. If S represents a non-priced asset then either form (B) or (C) may be used, depending on whether it is easier to determine appropriate values for θ or for ρ.

11.1.4 Examples

Example: Optimal Renewable Resource Extraction
Pindyck (1984) discusses the optimal extraction rate and in situ rents of a renewable resource. Suppose that the stock of a resource, S, is governed by the controlled stochastic process

$$dS = (B(S) - q)dt + \sigma S dz,$$

where $B(S)$ is a biological growth function and q is the harvest rate of the resource. Typically, there will be a value, K, such that $B(K) = 0$. Also we require that $B(0) = 0$, so 0 is an absorbing barrier of the process (there is

no return from extinction) and that $B' > 0$ for $S \in (0, K)$ and $B' < 0$ for $S > K$. K can be thought of as an environmental carrying capacity; the resource tend to shrink once it becomes greater than K.

Suppose that marginal costs depend only on the stock of the resource with the specific functional form

$$C(q) = c(S)q.$$

The total surplus (consumer plus producer) is

$$f(S, q) = \int_0^q D^{-1}(z)dz - c(S)q$$

With a discount rate of ρ, the Bellman Equation for this optimization problem is

$$\rho V = \max_q \int_0^q D^{-1}(z)dz - c(S)q + (B(S) - q)\, V_S + \tfrac{1}{2}\sigma^2 S^2 V_{SS}.$$

The FOC for the optimal choice of q is

$$D^{-1}(q) - c(S) - V_S(S) = 0,$$

or

$$q^* = D(c(S) + V_S).$$

Notice that the FOC implies that marginal surplus of an additional unit of the harvested resource is equal to the marginal value of an additional unit of the in situ stock:

$$f_q\,(S, q^*) \equiv D^{-1}(q^*) - c(S) = V_S(S).$$

To make further progress towards specific solutions we must parameterize the demand, cost and growth functions. Following Pindyck, we assume that the demand for the harvested resource is iso-elastic,

$$q = D(p) = bp^{-\eta},$$

the cost function is

$$c(S) = cS^{-\gamma},$$

and the biological growth function is of the form

$$B(S) = \frac{\alpha}{\beta} S \left(1 - \left(\frac{S}{K}\right)^{\beta}\right)$$

(the limiting case as $\beta \to 0$ is $B(S) = \alpha S \ln(K/S)$). The biological growth function is mean reverting with $S < K$ resulting in expected increases in the stock and $S > K$ resulting in expected decreases in the stock. The parameters α and β determine the speed of the mean reversion. Other things being equal, increasing β causes the size of positive changes to be greater and negative changes to be smaller when the stock is far from K (for stock levels near K the value of β has little effect). These features are illustrated in Figure 11.1, which shows the mean function for alternative values of β (shown here with $\alpha = 0.05$ and $K = 1$). Features of the this process are discussed further in Section 10.2.2 on page 304. [Note that the mean growth rate is maximized at $S = (1+\beta)^{-1/\beta}$, which goes to $1/e$ as $\beta \to 0$]. The model is summarized in Example Box 1.

The concentrated Bellman's Equation using these functional forms becomes[7]

$$\rho V = -\frac{b}{1-\eta}\left(cS^{-\eta} + V_S\right)^{1-\eta} + \frac{\alpha}{\beta} S \left(1 - (S/K)^{\beta}\right) V_S + \tfrac{1}{2}\sigma^2 S^2 V_{SS}. \quad (11.8)$$

The boundary conditions for the problem require that the marginal surplus must become infinite at $S = 0$ and be zero as S goes to infinity:

$$\lim_{S \to 0} V_S(S) = \infty \text{ and } \lim_{S \to \infty} V_S(S) = 0.$$

As the stock gets small and hence the catch rate declines, the value of an additional unit of the resource becomes infinitely high because of the form of the demand function. On the other hand, as the stock gets very large, an additional unit of the resource has no value at all because, after a point, the harvest of additional units does not justify the cost and those units merely contribute to the crowding and hence the mortality of the resource.

Differential equations of the form (11.8) generally have no closed-form solution and therefore it is necessary to solve them numerically. In the special

[7]Following Pindyck, we ignore the technicality that the integral defining the surplus does not converge when the lower limit of integration is 0. The lower limit is treated as a constant and ignored.

Example Box 11.1: Optimal Harvest of a Renewable Resource

Problem:

$$\max_q \int_0^\infty e^{-\rho t} f(S,q)dt$$

s.t.

$$dS = (B(S) - q)dt + \sigma S dz$$

Variables:

q	harvest (consumption) rate (control: quantity per period)
S	resource stock (state: quantity)

Parameters:

$B(S)$	biological growth function: $B(S) = \alpha S(1 - (S/K)^\beta)/\beta$
$D(p)$	demand function: $D(p) = bp^{-\eta}$
$c(S)$	marginal cost function: $c(S) = cS^{-\gamma}$
$f(S,q)$	surplus function: $f(S,q) = D^{-1}(z)dz - c(S)q$
ρ	discount rate

with $B(0) = 0$, $B(K) = 0$, $B'(S) > 0$ for $0 < S < K$ and $B'(S) < 0$ for $S > K$

Bellman's Equation:

$$\begin{aligned}
\rho V &= \max_q \int_0^q D^{-1}(z)dz - c(S)q + (B(S) - q)\, V_S + \tfrac{1}{2}\sigma^2 S^2 V_{SS} \\
&= \max_q \frac{b^{1/\eta}}{1 - 1/\eta} q^{1-1/\eta} - cS^{-\gamma} q \\
&\qquad + \left(\frac{\alpha}{\beta} S\left(1 - (S/K)^\beta\right) - q\right) V_S + \tfrac{1}{2}\sigma^2 S^2 V_{SS}
\end{aligned}$$

Boundary Conditions:

$$V_S(0) = \infty$$

$$V_S(\infty) = 0$$

Optimal Harvest Function:

$$q^* = b(cS^{-\gamma} + V_S)^{-\eta}.$$

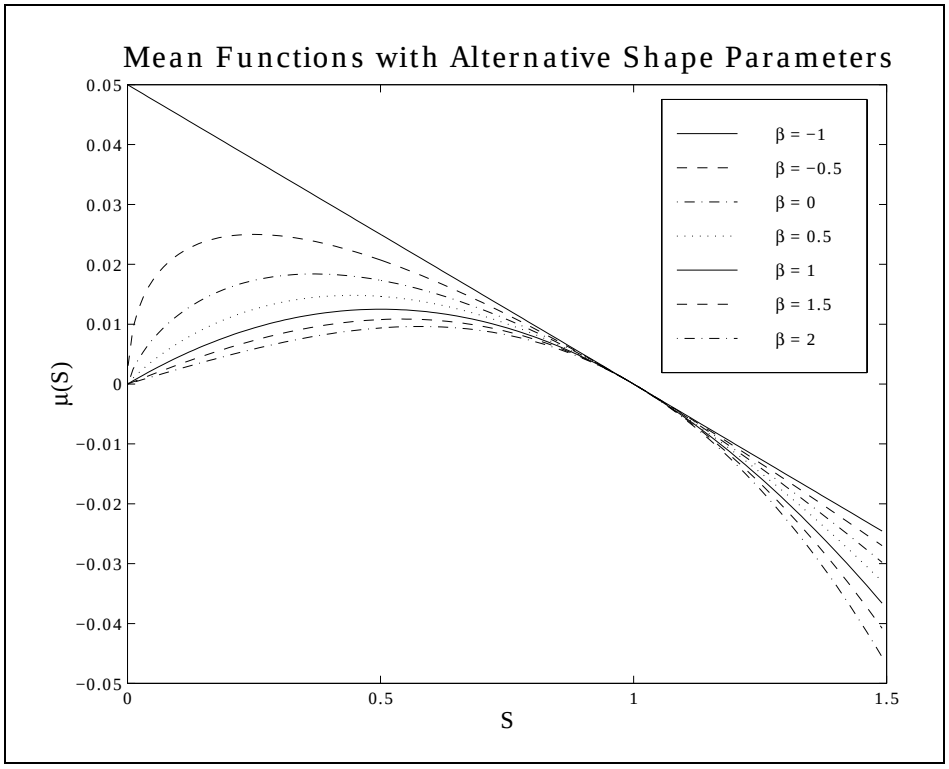

Figure 11.1

case that $\gamma = 1 + \beta$ and $\eta = 1/(1 + \beta)$, however, an analytic solution is possible. Specifically,

$$V(S) = -\phi\left(\frac{1}{S^\beta} + \frac{\alpha}{\rho K^\beta}\right)$$

where ϕ solves

$$\theta\phi^{\frac{1+\beta}{\beta}} - \beta\phi - c = 0,$$

and

$$\theta = \left(\frac{\alpha+\rho}{b}\frac{\beta}{1+\beta} - \frac{\beta\sigma^2}{2b}\right)^{\frac{1+\beta}{\beta}}.$$

It is straightforward to solve for ϕ using a standard root finding solver (see Section ??) and for some values of β an explicit solution is possible. Table

11.1 provides three special cases discussed by Pindyck that have closed form solutions, including the limiting case as $\beta \to 0$.

Table 11.1: Known Solutions to the Optimal Harvesting Problem

β	η	γ	$B(S)$	$V(S)$	$V_S(S)$	$q^*(S)$
1	$\frac{1}{2}$	2	$\alpha S(1-S/K)$	$-\phi_1\left(\frac{1}{S}+\frac{\alpha}{\rho K}\right)$	$\frac{\phi_1}{S^2}$	$\sqrt{c+\phi_1}S$
0	1	1	$\alpha S\ln(K/S)$	$\frac{b}{\alpha+\rho}\ln(S)+\phi_2$	$\frac{b}{(\alpha+\rho)S}$	$\frac{b(\alpha+\rho)}{b+(\alpha+\rho)c}S$
$-\frac{1}{2}$	2	$\frac{1}{2}$	$2\alpha S\left(\sqrt{\frac{K}{S}}-1\right)$	$-\phi_3\left(\sqrt{S}+\frac{\alpha\sqrt{K}}{\rho}\right)$	$-\frac{\phi_3}{\sqrt{S}}$	$\frac{b}{(c-\frac{1}{2}\phi_3)^2}S$

where

$$\begin{aligned}
\phi_1 &= 2\left(\frac{b}{\alpha+\rho-\sigma^2}\right)^2\left(1+\sqrt{1+c\left(\frac{\alpha+\rho-\sigma^2}{b}\right)^2}\right)\\
\phi_2 &= \frac{1}{\rho}\left(\left(\ln\left(\frac{b(\alpha+\rho)}{1+(\alpha+\rho)c}\right)-1\right)b+\frac{\alpha\ln(K)-\frac{1}{2}\sigma}{\alpha+\rho}\right)\\
\phi_3 &= c-\sqrt{c^2+\frac{2b}{\alpha+\rho+\sigma^2/8}}
\end{aligned}$$

Example: Stochastic Growth

Cox, Ingersoll and Ross (1985) develop a growth model in which the productivity of capital, K, depends, both in mean and variance, on an exogenous technology shock, denoted Y. Y is governed by

$$dY = (aY-b)dt + \sigma\sqrt{Y}dz.$$

With c denoting current consumption (the control), the capital stock dynamics are

$$dK = (\beta KY - c)dt + \epsilon K\sqrt{Y}dz,$$

where the same Brownian motion, dz, that drives the technology shocks also causes volatility in the productivity of capital. The social planner's optimization problem is to maximize the present value of the utility of consumption,

taken here to be the log utility function, using discount rate ρ. The problem is summarized in Example Box 2.

Example Box 11.2: Optimal Growth

Problem:

$$\max_c \int_0^\infty e^{-\rho t} \ln(c) dt$$

s.t.

$$dY = (aY - b)dt + \sigma\sqrt{Y}dz$$

$$dK = (\beta KY - c)dt + \epsilon K\sqrt{Y}dz$$

Variables:

c	consumption rate (control: quantity per period)
Y	technology shock (state)
K	capital stock (state: quantity)

Parameters:

$\ln(c)$	utility function
a, b, σ	technology shock dynamics
β, ϵ	capital productivity parameters
ρ	discount rate

Bellman's Equation:

$$\begin{aligned} \rho V = \max_c \ln(c) &+ V_K(\beta KY - c) + V_Y(aY - b) \\ &+ \tfrac{1}{2}V_{KK}\epsilon^2 K^2 Y + \tfrac{1}{2}V_{YY}\sigma^2 Y + V_{KY}\sigma\epsilon KY \end{aligned}$$

Optimal Consumption: $c^* = \rho K$

Before discussing the solution it is useful to consider the form of the technology assumed here. The expected growth rate in capital, ignoring consumption, is affine in the capital stock and depends on the size of the technology shock. The technology shock, in turn, has an expected growth

pattern given by

$$dEY = (aEY - b)dt.$$

This differential equation can be solved for the expected value of Y:[8]

$$E_0 Y_t = (Y_0 - b/a)e^{at} + b/a.$$

Roughly speaking, this implies that, for a given capital stock, the productivity of capital is expected to grow at a constant rate (a) if Y is greater than b/a and to shrink at the same rate when Y is less than b/a (the model might make more sense if we take a and b to be negative parameters; this would imply that b/a is a stable point rather than an unstable one).

Let us guess that the solution is one with consumption proportional to the capital stock

$$c = \alpha K.$$

The FOC condition associated with the Bellman equation (see Example Box 2) tells us that the optimal c satisfies

$$1/c = V_K.$$

If our guess is right it implies that $V(K, Y) = \ln(K)/\alpha + f(Y)$, where $f(Y)$ is yet to be determined. To verify that this guess is correct, substitute it into the Bellman equation:

$$\rho\left(\frac{\ln(K)}{\alpha} + f(Y)\right) = \ln(\alpha K) + \left(\frac{\beta}{\alpha}Y - 1\right) + f'(Y)(aY - b) - \frac{\epsilon^2 Y}{2\alpha} + \tfrac{1}{2}f''(Y)\sigma^2 Y.$$

Collecting terms and simplifying, we see that $\alpha = \rho$ and that $f(Y)$ solves a certain second order differential equation.

Rather than try to solve $f(Y)$ directly, however, a more instructive approach is to solve for the value function directly from its present value form. If our guess is correct then

$$V(K, Y) = E\left[\int_0^\infty e^{-\rho t}\ln(\rho K)dt\right] = \frac{\ln(\rho)}{\rho} + \int_0^\infty e^{-\rho t}E\left[\ln(K)\right]dt \quad (11.9)$$

[8] This is the same functional form as equation 10.3 on 290.

The only difficulty presented here is to determine the time path of $E[\ln(K)]$. Using Ito's Lemma and $c = \rho K$

$$d\ln(K) = \frac{dK}{K} - \tfrac{1}{2}\epsilon^2 Y dt = \left[\left(\beta - \tfrac{1}{2}\epsilon^2\right) Y - \rho\right] dt + \sigma\sqrt{Y} dz.$$

Taking expectations and using the previously obtained result for EY yields

$$\begin{aligned} dE[\ln(K)] &= \left[\left(\beta - \tfrac{1}{2}\epsilon^2\right) E[Y] - \rho\right] dt \\ &= \left[\left(\beta - \tfrac{1}{2}\epsilon^2\right)\left(\left(Y_0 - \tfrac{b}{a}\right) e^{at} + \tfrac{b}{a}\right) - \rho\right] dt \\ &= \left[c_0 a e^{at} + c_1\right] dt, \end{aligned}$$

where

$$c_0 = \frac{\beta - \frac{1}{2}\epsilon^2}{a}\left(Y_0 - \frac{b}{a}\right)$$

$$c_1 = \frac{b}{a}\left(\beta - \tfrac{1}{2}\epsilon^2\right) - \rho.$$

Integrating both sides and choosing the constant of integration to ensure that, at $t = 0$, the expected value of $E[\ln(K_t)] = \ln(K_0)$ produces an expression for $E[\ln(K)]$ when $c = \rho K$:

$$E[\ln(K)] = \ln(K_0) - c_0 + c_0 e^{at} + c_1 t.$$

One step remains; we must use the formula for $E[\ln(K)]$ to complete the derivation of the present value form of the value function. Recalling (11.9)[9]

$$\begin{aligned} V(K,Y) &= \int_0^\infty e^{-\rho t} E\left[\ln(K)\right] dt + \frac{\ln(\rho)}{\rho} \\ &= \int_0^\infty \left(\left(\ln(K_0) - c_0\right) e^{-\rho t} + c_0 e^{(a-\rho)t} + c_1 t e^{-\rho t}\right) dt + \frac{\ln(\rho)}{\rho} \\ &= \frac{\ln(K_0) - c_0}{\rho} + \frac{c_0}{\rho - a} + \frac{c_1}{\rho^2} + \frac{\ln(\rho)}{\rho}. \end{aligned}$$

[9]If the third line is problematic for you it might help to note that

$$\int t e^{-\rho t} dt = -\frac{e^{-\rho t}}{\rho}\left(t + \frac{1}{\rho}\right).$$

Substituting in the values of c_0 and c_1 and rearranging we obtain an expression for the value function

$$V(K,Y) = \frac{\ln(K)}{\rho} + \frac{\beta - \frac{1}{2}\epsilon^2}{\rho(\rho - a)}Y + \frac{1}{\rho}\left(\ln(\rho) - \frac{b(\beta - \frac{1}{2}\epsilon^2)}{\rho(\rho - a)} - 1\right).$$

(the subscripts on K and Y are no longer necessary). Notice that this does indeed have the form $\ln(K)/\rho + f(Y)$, with $f(Y)$ a linear function of Y. We have therefore satisfied the essential part of Bellman's equation, namely verifying that $c = \rho K$ is an optimal control. We leave as an exercise the task of completing the verification that Bellman's equation is satisfied by our expression for $V(K,Y)$.

Let's review the steps we took to solve this problem. First, we guessed a solution for the control and then used the first order conditions from Bellman's equation to determine a functional form for $V(K,Y)$ that must hold for this to be an optimal control. We then evaluated the present value form of the value function for this control, thereby obviating the need to worry about the appropriate boundary conditions on Bellman's equation (which we have seen is a delicate subject). We were able to obtain an expression for the value function that matched the functional form obtained using the first order conditions, verifying that we do indeed have the optimal control. This strategy is not always possible, of course, but when it is, we might as well take advantage of it.

Example: Portfolio Choice

The previous examples had a small number of state and control variables. In the example we are about to present, we start out with a large number of both state variables and controls, but with a specific assumption about the state dynamics, the dimension of the state is reduced to one and the control to two. Such a reduction makes a problem from that is essentially impossible to solve in general into one that is much closer to being solved. If a specific class of reward functions is used, the problem can be solved explicitly (we leave this as an exercise).

Suppose investors have a set of n assets from which to invest, with the per unit price of these assets generated by an n dimensional Ito process

$$dP = \mu(P)dt + \sigma(P)dz,$$

Example Box 11.3: Portfolio Choice

Problem:

$$\rho V = \max_{C,w} U(C) + \left[Ww^\top \mu - C\right] V_W + \tfrac{1}{2} W^2 w^\top \Sigma w V_{WW},$$

s.t. $\sum_i w_i = 1$ and

$$dP = \mu P dt + \sigma dz$$

$$dW = \left[Ww^\top \mu - C\right] dt + Ww^\top \sigma dz.$$

$$w_i = N_i P_i / W.$$

Variables:

p	price (underlying n-dimensional state)
W	wealth (state)
C	consumption (control)
w	share of weath in asset i (n-dimensional control)

Optimality Conditions:

$$U'(C) = V_W,$$

$$WV_W \mu + W^2 V_{WW} \Sigma w - \lambda \underline{1} = 0,$$

and

$$\sum_i w_i = 1,$$

Solution:

$$w = \nu + \alpha(W)\theta, \tag{11}$$

where

$$\nu = \frac{\Sigma^{-1}\underline{1}}{\underline{1}^\top \Sigma^{-1} \underline{1}},$$

$$\theta = \Sigma^{-1}\left(\mu - \frac{\underline{1}^\top \Sigma^{-1} \mu}{\underline{1}^\top \Sigma^{-1} \underline{1}} \underline{1}\right)$$

and

$$\alpha(W) = -\frac{V_W}{WV_{WW}},$$

where $\sigma(P)$ is an $n \times k$ matrix valued function (i.e., $\sigma : \Re^n \to \Re^{n \times k}$), and dz is a k-dimensional vector of independent Wiener processes. We assume that $\Sigma = \sigma\sigma^\top$, the instantaneous covariance matrix for prices, is non-singular, implying that there are no redundant assets or, equivalently, that there is no riskless asset.[10] A portfolio can be defined in number of shares, N_i, invested in each asset or as the fraction of wealth held in each asset:

$$w_i = N_i P_i / W.$$

Expressed in terms of N_i the wealth process can be described by

$$dW = \sum_{i=1}^{n} N_i dP_i$$

whereas in terms of w_i it is given by

$$dW/W = \sum_{i=1}^{n} w_i dP_i / P_i.$$

The latter expression is particularly useful if prices are multivariate geometric Brownian motion processes, so $\mu(P)$ and $\sigma(P)$ are constants μ and σ, implying that:

$$dW/W = w^\top \mu dt + w^\top \sigma dz,$$

i.e., that W is itself a geometric Brownian motion process. This means that portfolio decisions can be expressed in terms of wealth alone, without reference to the prices of the underlying assets in the portfolio. Geometric Brownian motion, therefore, allows for a very significant reduction in the dimension of the state (from n to 1).

Consider an investor who draws off a flow of consumption expenditures C. The wealth dynamics are then

$$dW = \left[W w^\top \mu - C\right] dt + W w^\top \sigma dz.$$

Suppose the investor seeks to maximize the discounted stream of satisfaction derived from consumption, where utility is given by $U(C)$ and the discount rate is ρ. The Bellman's Equation for this problem is[11]

$$\rho V = \max_{C,w} U(C) + \left[W w^\top \mu - C\right] V_W + \tfrac{1}{2} W^2 w^\top \Sigma w V_{WW},$$

[10] the case in which a riskless asset is available is treated in an exercise.

[11] If prices were not geometric Brownian motion the coefficients μ and σ would be functions of current prices and the Bellman's Equation would have additional terms representing derivatives of the value function with respect to prices, which would make the problem considerably harder to solve.

s.t. $\sum_i w_i = 1$.

The FOC associated with this maximization problem are

$$U'(C) = V_W, \tag{12a}$$

$$WV_W\mu + W^2V_{WW}\Sigma w - \lambda\underline{1} = 0, \tag{12b}$$

and

$$\sum_i w_i = 1, \tag{12c}$$

where λ is a Lagrange multiplier introduced to handle the adding-up constraint on the w_i. A bit of linear algebra applied to (12b) and (12c) will demonstrate that the optimal portfolio weight vector, w, can be written as a linear combination of vectors, ν and θ, that are independent of the investor's preferences:

$$w = \nu + \alpha(W)\theta, \tag{13}$$

where

$$\nu = \frac{\Sigma^{-1}\underline{1}}{\underline{1}^\top\Sigma^{-1}\underline{1}},$$

$$\theta = \Sigma^{-1}\left(\mu - \frac{\underline{1}^\top\Sigma^{-1}\mu}{\underline{1}^\top\Sigma^{-1}\underline{1}}\underline{1}\right)$$

and

$$\alpha(W) = -\frac{V_W}{WV_{WW}},$$

This has a nice economic interpretation. When asset prices are generated by geometric Brownian motion, a portfolio separation result occurs, much like in the static CAPM model. Only two portfolios are needed to satisfy all investors, regardless of their preferences. One of the portfolios has weights proportional to $\Sigma^{-1}\underline{1}$, the other to $\Sigma^{-1}(\mu - (\nu^\top\mu)\underline{1})$. The relative amounts held in each portfolio depend on the investor's preferences, with more of the first portfolio being held as the degree of risk averse rises (for smaller values of $\alpha(W)$). This is understandable when it is noticed that the first portfolio is the minimum risk portfolio, i.e., ν solves the problem

$$\min_\nu \nu^\top\Sigma\nu, \text{ s.t. } \nu^\top\underline{1} = 1.$$

Furthermore, the expected return on the minimum risk portfolio is $\nu^\top \mu$; hence the term $\mu - (\nu^\top \mu)\underline{1}$ can therefore be thought of as an "excess" return vector, i.e., the expected returns over the return on the minimum risk portfolio.

The problem is therefore reduced to determining the two decision rule functions for consumption and investment decisions, $C(W)$ and $\alpha(W)$, that satisfy:

$$U'(C(W)) = V_W(W)$$

and

$$\alpha(W) = -\frac{V_W(W)}{W V_{WW}}.$$

Notice that the two fund separation result is a result of the assumption that asset prices follow geometric Brownian motions and not the result of any assumption about preferences. Given the enormous simplification that it allows, it is small wonder that financial economists like this assumption.

11.2 Free Boundary Problems

We have already seen how boundary conditions are needed to determine the solution to dynamic models in continuous time. Many important problems in economics, however, involve boundaries in the state space the location of which must be determined as part of the solution. Such problems are known as free boundary problems. The boundary will either mark the location where some discrete decision is made or will represent a location at which some transition takes place.[12]

Table 11.2 contains a classification of different free boundary problems that have appeared in the economics literature. The most important distinction, both in understanding the economics and in solving the problem numerically, is whether the boundary can be crossed. If the control is such that it maintains a stochastic process within some region defined by the free boundary, the problem is a barrier problem and we will solve a differential equation in this region only. For example, the stock of a stochastic renewable

[12]In the physical sciences free boundary problems are also known as Stefan problems. A commonly used example is the location of the phase change between liquid and ice, where the state space is measured in physical space coordinates.

resource can be harvested in such a way as to keep the stock level below some specified point. If the stock rises to this point, it is harvested in such a way as to maintain it at the boundary (barrier control) or to some point below the boundary (impulse control).

In barrier controls problems, the barrier defines a trigger point at which, if reached, one maintains the state at the barrier by exactly offsetting any movements across the barrier. Typically, such a control is optimal when there are variable costs associated with exerting the control. In such a situation it is only optimal to exert the control if the marginal change in the state offsets the marginal cost of exerting the control.

In impulse control problems, if the barrier is reached one takes an action that instantaneously moves the state to a point inside the barrier. An (s, S) inventory control system is an example of an impulse control in which the state is the level of inventory, which is subject to random demand. When the inventory drops to the level s, an order to replenish it to level S is issued. Typically such controls are optimal when there is a fixed cost associated with exerting the control; the control is exerted only when the benefit from exerting the control covers the fixed cost.

The other major type of free boundary problem arises when, in addition to one or more continuous state variables, there is also a state that can take on discrete set of values. In this case, boundaries represent values of the continuous states at which a change in the discrete state occurs. For example, consider a firm that can either be actively producing or can be inactive (a binary state variable). The choice of which state is optimal depends on a randomly fluctuating net output price. Two boundaries exist that represent the prices at which the firm changes from active to inactive or from inactive to active (it should be clear that the latter must be above the former to prevent the firm from having to be continuously changing!).

An important special case of the discrete state problem is the so-called optimal stopping problem; the exercise of an American option is perhaps the most familiar example. Stopping problems arise when the choice of one of the discrete state values is irreversible. Typically the discrete state takes on two values, active and inactive. Choosing the inactive state results in an immediate one time payout. An American put option, for example, can be exercised immediately for a reward equal to the option's exercise price less the price of the underlying asset. It is optimal to exercise when the underlying asset's price is so low that it is better to have the cash immediately and reinvest it than to wait in hopes that the price drops even lower.

Table 11.2: Types of Free Boundary Problems

BARRIERS:

Problem	Action	Boundary Condition	Optimality Condition
Impulse control	Jump from boundary	$V(S^\tau) = V(S^\tau + J^\tau) - F$	$V_S(S^*) = V_S(S^* + J^*) \pm mc$
Barrier control	Move along boundary	$V_S(S^\tau) = \pm mc$	$V_{SS}(S^*) = 0$

TRANSITIONAL BOUNDARIES:

Problem	Action	Boundary Condition	Optimality Condition
Discrete states	Change state	$V^i(S^\tau) = V^j(S^\tau) - F^{ij}$	$V_S^i(S^*) = V_S^j(S^*)$
Bang-bang	Switch between control extrema	$V^0(S^\tau) = V^1(S^\tau)$, $V_S^0(S^\tau) = V_S^1(S^\tau)$	$V_{SS}^0(S^*) = V_{SS}^1(S^*)$

Notes:
S^τ represents a point on an arbitrary boundary where an action is taken
S^* represents a point on an optimally chosen boundary where an action is taken
J^τ and J^* represent arbitrary and optimal jump sizes in impulse control
F and mc are fixed cost and marginal cost, respectively
In impulse and barrier control:
use $+mc$ if the boundary is approached from above
use $-mc$ if approached from below
(for cost minimization problems reverse the signs)
In the discrete state problem the switch is from state 0 to state 1

Another important special case is the so-called stochastic bang-bang problem. Such problems arise when it is optimal to exert a bounded continuous control at either its maximum or minimum level. Effectively, therefore, there is a binary state variable that represents which control level is currently being exerted. The free boundary determines the values of the continuous variables at which it is optimal to change the binary state.

A couple points should be mentioned now and borne in mind whenever considering free boundary problems. First, it is useful to distinguish between the value function of a problem at an arbitrary choice of a boundary and the optimal choice of the boundary. The value function (the present value of the return stream) using an arbitrary barrier control is described by a second order partial differential equation subject to the appropriate boundary conditions; this is the message of the Feynman-Kac equation (see Section 10.1.2). The optimal choice of the boundary must then add additional restrictions that ensure its optimality. We therefore distinguish in Table 11.2 between a point, S^τ, on an arbitrary boundary and a point, S^*, on the optimal boundary. As we shall see in the next chapter, this distinction is particularly important when using a strategy to find the free boundary that involves guessing its location, computing the value function for that guess, and evaluating a condition that should hold at the boundary.

Related to this is an understanding the number of boundary conditions that must be applied. Here are some rules that should help you avoid problems. First, any non-stochastic continuous state will have one partial derivative and will require one boundary condition. On the other hand, any stochastic state variable will have second order derivatives and will generally need two boundary conditions.[13] These statements apply to both arbitrary and optimal controls.

For optimality we will require an additional boundary condition for every parameter needed to define the control. Thus if the control is defined by a single point in the state space, we need one additional constraint to define the location of that point. If the control is a curve in a 2-dimensional state space, we will need a single functional constraint, that will take the form

$$b(S, V, V_S, V_{SS}) = 0$$

(this will become a lot clearer after you've read through some examples). The additional constraints can be derived formally by maximizing the value

[13]The exception to this rule of thumb involves processes that exhibit singularities at natural boundaries, which can eliminate the need to specify a condition at this boundary

function for an arbitrary barrier with respect to the location of the barrier, which for single points means solving an ordinary maximization problem and for functional barriers means solving an optimal control problem.

In all of these cases one can proceed as before by defining a Bellman's Equation for the problem and solving the resulting maximization problem. The main new problem that arises lies in determining the region of the state space over which the Bellman's Equation applies and what conditions apply at the boundary of this region. We will come back to these points so if they are not clear now bear with us. Now let us consider each of the main types of problem and illustrate them with some examples.

11.2.1 Impulse Control

Impulse and barrier control problems arise when the reward function includes the size of the change in a state variable caused by exerting some control. Such problems typically arise when there are transactions costs associated with exerting a control, in which case it may be optimal to exert the control at an infinite rate at discrete selected times. In addition, the reward function need not be continuous in ΔS.

The idea of an infinite value for the control may seem puzzling at first and one may feel that it is unrealistic. Consider that in many applications encountered in economics the control represents the rate of change in a state variable. The state is typically a stock of some asset measured in quantity units. The control is thus a flow rate, measured in quantity units per unit time. If the control is finite, the state cannot change quickly; essentially the size of the change in the state must grow small as the time interval over which the change is measured gets small.

In many situations, however, we would like to have the ability to change the state very quickly in relation to the usual time scale of the problem. For example, the time it takes to cut down a timber stand may be very small in relation to the time it takes for the stand to grow to harvestable size. In such situations, allowing the rate of change in the state to become infinite allows us to change the state very quickly (instantaneously). Although this makes the mathematics somewhat more delicate, it also results in simpler optimality conditions with intuitive economic interpretations.

Consider the single state case in which the state variable governed by

$$dS = [\mu(S) + x]dt + \sigma(S)dz$$

and the reward function that is subject to fixed and variable costs associated with exerting the control:

$$f(S, \Delta S, x) = \begin{cases} r^-(S) - c^-(-\Delta S) - F^- & \text{if } x < 0 \\ r^0(S) & \text{if } x = 0 \\ r^+(S) - c^+(\Delta S) - F^+ & \text{if } x > 0 \end{cases}$$

with $c^-(0) = c^+(0) = 0$. In this formulation there are fixed costs, F^- and F^+, and variable costs, c^- and c^+, associated with exerting the control, both of which depend on the sign of the control. Typically, we would assume that the fixed costs are positive. The variable costs, however, could be negative; consider the salvage value from selling off assets. To rule out the possibility of arbitrage profits (when the reward is increasing in the state: $r_S \geq 0$), we require that

$$F^+ + c^+(z) + F^- + c^-(-z) > 0$$

for any positive z; thereby preventing infinite profits to be made by continuous changes in the state.

With continuous time diffusion process, which are very wiggly, any strategy that involved continuous readjustment of a state variable would become infinitely expensive and could not be optimal. Instead the optimal strategy is to change the state instantly in discrete amounts, thereby incurring the costs of those states only at isolated instants of time. An impulse control strategy would be optimal when there are non-zero fixed costs ($F^+, F^- > 0$). Barrier control strategies (which we discuss in the next section) arise when the fixed cost components of altering the state are zero.

With impulse control, the state of the system is reset to a new position (a target) when a boundary is reached (a trigger). It may be the case that either or both the trigger and target points are endogenous and need to be determined. For example, in a cash management situation, a bank manager must determine when there is enough cash-on-hand (the trigger) to warrant investing some of it in an interest bearing account as well as how much cash to retain (the target). Alternatively, in an inventory replacement problem, an inventory is restocked when it drops to zero (the trigger), but the restocking level (the target) must be determined (restocking occurs inatantaneously so there is no reason not to let inventory fall to zero). A third possability arises in an asset replacement problem, where the age at which an old machine is replaced by a new one must be determined (the trigger), but the target is known (the age of a new asset).

In any impulse control problem, a Feynman-Kac Equation governs the behavior of the value function on a region where control is not being exerted. The boundaries of the region are determined by value matching conditions that equate the value at the trigger point with the value at the target point less the cost of making the jump. Furthermore, if the trigger is subject to choice, a smooth pasting condition is imposed that the marginal value of changing the state is equal to the marginal cost of making the change. A similar condition holds at the target point if it is subject to choice. For those wishing a rigorous discussion and verification of these points see Appendix A.

Example: Asset Replacement (Cows revisited)
In Chapter ?? we examined a discrete time and state problem concerning the optimal age to replace an asset. The specific example involved the number of production cycles after which a milk cow should be replaced by a one-year old cow. The value of the cow depends on her current and future yield potential, which is described by $y(A)$, where A is the state variable representing the age of the cow. The value also depends on the net price of milk, P, and the net cost of replacing the cow, c.

This is a deterministic problem in which the state dynamics are simply $dA = dt$. The reward function is $y(A)P$. Thus the Bellman equation is

$$\rho V(A) = y(A)P + V'(A).$$

This differential equation is solved on the range $A \in [1, A^*]$, where A^* is the optimal replacement age. The boundary conditions are given by the value matching condition:

$$V(1) = V(A^*) + c$$

and the optimality (smooth pasting) condition:

$$V'(A^*) = 0$$

The smooth pasting condition may not be obvious, but it is intuitively reasonable if one considers that a cow above the age A^* should always be immediately replaced. Once past the age of A^*, therefore, the value function is constant: $V(A) = V(A^*) = V(1) - c$, for $A \geq A^*$. Also, no optimality condition is imposed at the lower boundary ($A = 1$) because we are not free to pick the age of the new cow.

The Bellman equation is a first order linear differential equation and hence can be solved analytically. In the specific case that $y(A)$ is quadratic in A the Bellman equation can be written as

$$V' = \rho V - P(a_0 + a_1 A + a_2 A^2)$$

which has solution

$$V(A) = ke^{\rho A} + \beta_0 + \beta_1 A + \beta_2 A^2,$$

where k is a constant to be determined by the boundary conditions and the β_i can be verified to satisfy the recursive conditions[14]

$$\beta_2 = Pa_2/\rho$$

$$\beta_i = Pa_i/\rho + (i+1)\beta_{i+1}/\rho,\ i = 1, 0.$$

To compute the values of the constant of integration, k, and the optimal replacement age A^* we impose the value matching and smooth pasting conditions. Although there are two unknowns here (k and A^*) it is easy to eliminate the k term:

$$k = \frac{\sum_{i=0}^{n} \beta_i \left(A^{*^i} - 1\right) + c}{e^{\rho} - e^{\rho A^*}}$$

to obtain the single root condition

$$\rho e^{\rho A}\left(\sum_{i=1}^{n} \beta_i (A^i - 1) + c\right) + \left(e^{\rho} - e^{\rho A}\right)\sum_{i=1}^{n} i\beta_i A^{i-1} = 0,$$

which can be solved using any univariate root finding algorithm (see Section ??).

Figure 11.2 displays the value function, with the star representing A^*. For values above A^* the value function is flat: a cow that old would be immediately replaced by a one year old and hence the value function equals $V(1)$ less the replacement cost c. The dashed curve for values $A > A^*$ represents the continuation of the value function but has no meaningful interpretation. It is included to make clear that $V(A)$ reaches a minimum at A^*, and therefore $V'(A^*) = 0$.

[14] It is straightforward to verify that if $y(A)$ were an nth order polynomial, the solution would have the same form with

$$\beta_n = Pa_n/\rho \text{ and } \beta_i = Pa_i/\rho + (i+1)\beta_{i+1}/\rho \text{ for } i = n-1, .., 0.$$

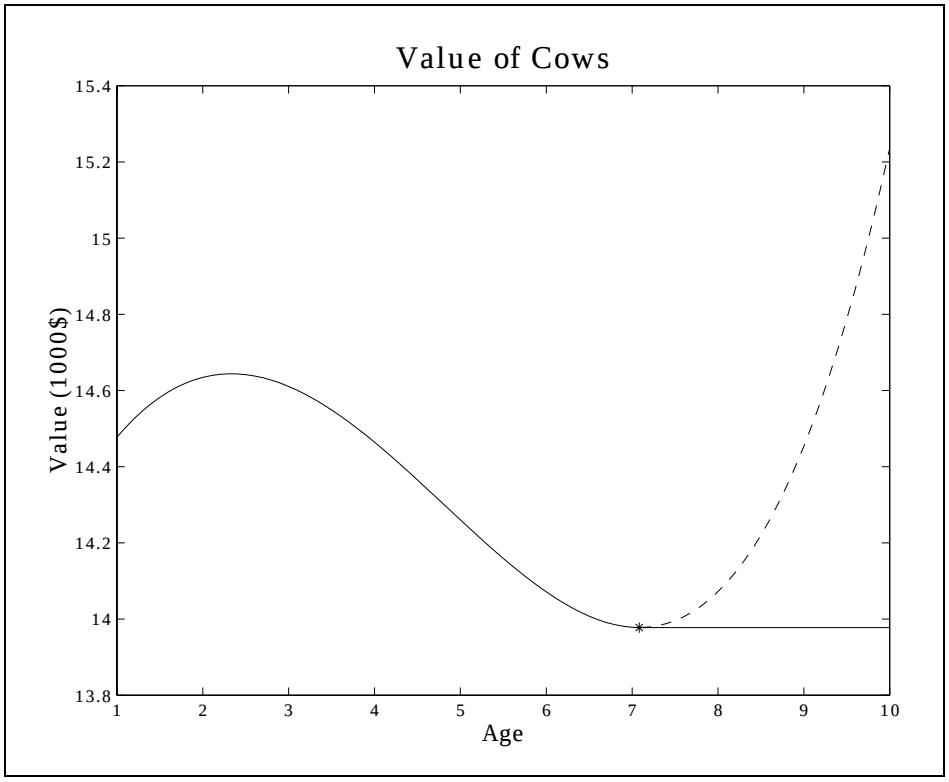

Figure 11.2

Before leaving this example, a potentially misleading interpretation should be discussed. Although we have referred to the value function as representing the value of a cow of age A, this is not quite correct. The value function represents the value of the assets used to milk a cow of age A. The distinction lies in the fact that the particular cow will be replaced at age A^*, at which point a new cow will be used. The current cow has value equal to the discounted stream of returns it generates:

$$\int_0^{A^*-A} e^{\rho t} P y(A+t) dt,$$

but the value function is

$$V(A) = \int_0^{A^*-A} e^{-\rho t} P y(A+t) dt + e^{-\rho(A^*-A)} V(A^*)$$

Thus the cow at age A has value

$$V(A) - e^{-\rho(A^*-A)}V(A^*).$$

Example: Timber Harvesting

In the previous example we examined an asset replacement problem in which the asset generated a continuous stream of net returns. In some cases, however, the returns are generated only at the replacement time. Consider a forest stand that will be clear-cut on a date set by the manager. The stand is allowed to grow naturally at a biologically determined rate according to

$$dS = \alpha(m - S)dt.$$

The parameter m represents a biological equilibrium point. When the stand is cut, it is sold for a net return of pS. In addition, the manager incurs a cost of c to replant the stand, which now has size $S = 0$. The decision problem is to determine the optimal cutting/replanting stand size, using a discount rate of ρ. The Bellman equation is

$$\rho V = \alpha(m - S)V'(S),$$

for $S \in [0, S^*]$, where S^* is determined by boundary conditions

$$V(S^*) = V(0) + pS^* - c \quad \text{value matching}$$

and

$$V'(S^*) = p \qquad \text{smooth pasting.}$$

If the stand starts at a size above S^* it is optimal to cut/replant immediately. Clearly the marginal value of additional timber when $S > S^*$ is the net return from the immediate sale of an additional unit of timber. Hence, for $S > S^*$, $V(S) = V(S^*) + p(S - S^*)$ and $V'(S) = p$.

The Bellman equation can be rewritten in the form

$$V' = \frac{\rho}{\alpha}\frac{V}{m - S},$$

the solution to which is easily verified to be

$$V = k(m - S)^{-\rho/\alpha},$$

where k is a constant of integration to be determined by the boundary conditions. There are, therefore, two unknowns to be determined, k and S^*. The value matching condition allows us to solve for k in terms of S^*:

$$k\left(S^*\right) = \frac{pS^* - c}{\left(m - S^*\right)^{-\rho/\alpha} - m^{-\rho/\alpha}}$$

The optimal S^* can be found using the smooth-pasting condition:

$$\frac{\rho}{\alpha p} k(S^*) - (m - S^*)^{\rho/\alpha + 1} = 0,$$

which can be solved with any one-dimensional root finding solver (see Section ??).

The method is illustrated in Code Box 2. This code sets $\alpha = p = m = 1$, $c = 0.15$ and $\rho = \frac{1}{2}$. The optimal harvest stand size is 0.4291, indicating that it is optimal to harvest the stand when it is less than one half its mature size (m). The value function for this problem is shown in Figure 11.3. The starred point represents the optimal harvest size. For stand sizes above this point the value function is linear. The dashed lines extend the two pieces of the value function beyond their domains to illustrate the value matching and smooth pasting conditions.

Code Box 11.2: Timber Harvesting

It is convenient to normalize by setting $\alpha = p = m = 1$, which amounts to picking scales for time, stand size and money. When the parameters thus normalized, the economically relevant range for the cost parameter is $[0, 1]$; for values greater than 1 it would never be optimal to harvest because the revenues thus generated would not cover the replanting costs.[15] The discount rate can be greater or less than the maximal timber growth rate, α , however. Comparative static exercises, therefore, need only examine the problem for normalized parameter values of (ρ, c) on the interval $[0, \infty) \times [0, 1]$.[16] Figure 11.4 is a contour plot of S^* illustrating the behavior of the optimal harvest stand size (as a fraction of the carrying capacity, m. It can be shown that as ρ/α gets large (i.e., either the future is heavily discounted or the timber grows very slowly), the optimal harvest size declines, reaching a lower bound

[15]This may not be true if the initial stand is larger than m, in which case it is optimal

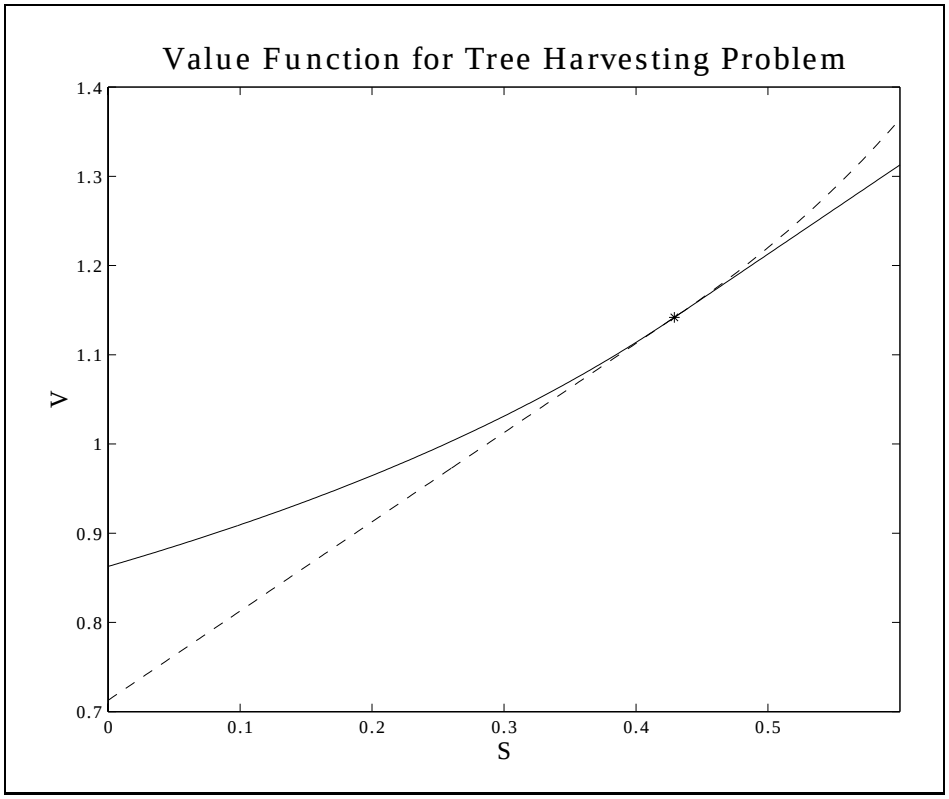

Figure 11.3

of c/p. This lower bound represents the point at which the returns from cutting just offset the cost of replanting. In this case the value function becomes identically zero.

On the other hand, the optimal harvest size increases as c/p increases. Intuitively, when replanting costs are high, it pays to delay harvest. Similarly, as timber revenue increases, it pays to cut sooner. Notice that, for $c = 0$, we

to harvest immediately if $S > c/p$.

[16]With some algebraic manipulation one can show that, for the normalized problem,

$$V(0) = (1 - S^*)^\rho V(S^*)$$

and

$$V(S^*) = (1 - S^*)/\rho.$$

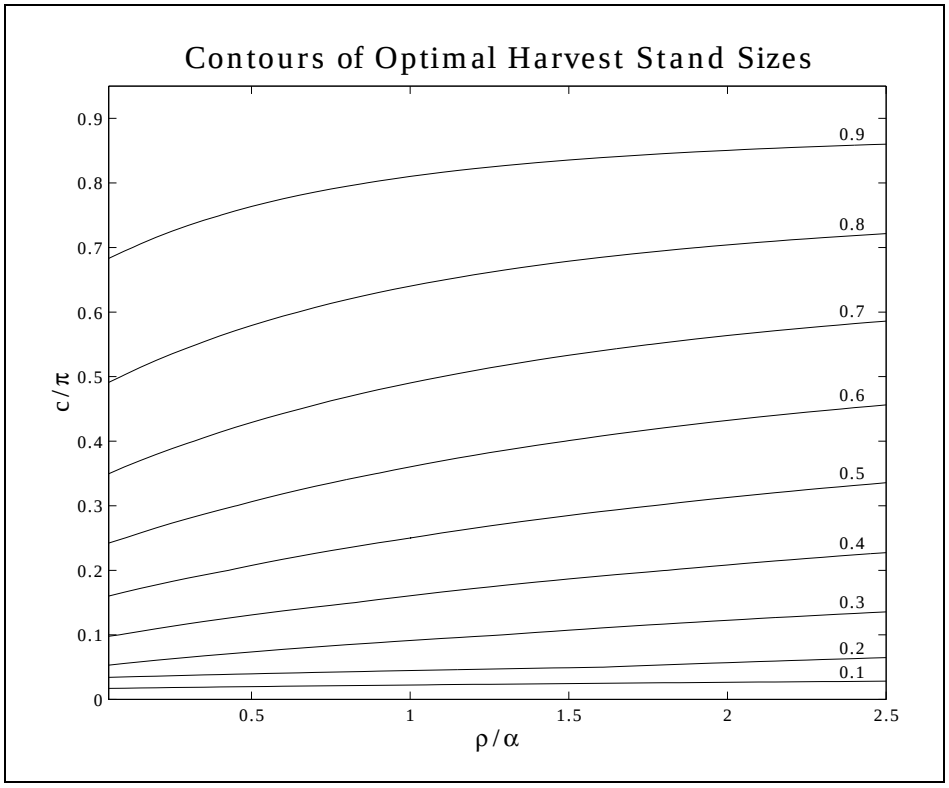

Figure 11.4

get the curious result that $S^* = 0$, i.e., that timber is harvested as soon as it is planted. In this case the value function is equal to $p\alpha m/\rho$. Essentially, when replanting costs are zero, one harvests continuously a crop that grows at rate αm; the discounted present value of this continuous harvesting is thus $1/\rho$ times the instantaneous return $p\alpha m$.

The timber harvesting problem with replanting is known as the Faustmann problem. A related problem, known as the Fisher problem, is the determination of the optimal harvest time when the stand is abandoned after being clear-cut (i.e., it not replanted). The Fisher problem is an optimal stopping problem and it solution differs only in the boundary conditions. The smooth-pasting condition is, in fact, the same but the value matching condition is simply

$$V(S^*) = pS^*.$$

We leave as an exercise the verification that the solution in this case is

$$S^* = \left(1 - \frac{1}{1+\rho/\alpha}\right) m$$

For the parameters used in the numerical example, this yields an optimal cutting size of 2/3, which is larger than the optimal cutting size when replanting will occur. A comparison of the value functions are shown in Figure 11.5. With these parameters, the value of the stand with abandonment is less than half the value with replanting, indicating that replanting is the preferred management mode in this case. With other parameter values, however, this need not be true, especially if the replanting cost, c, is high.

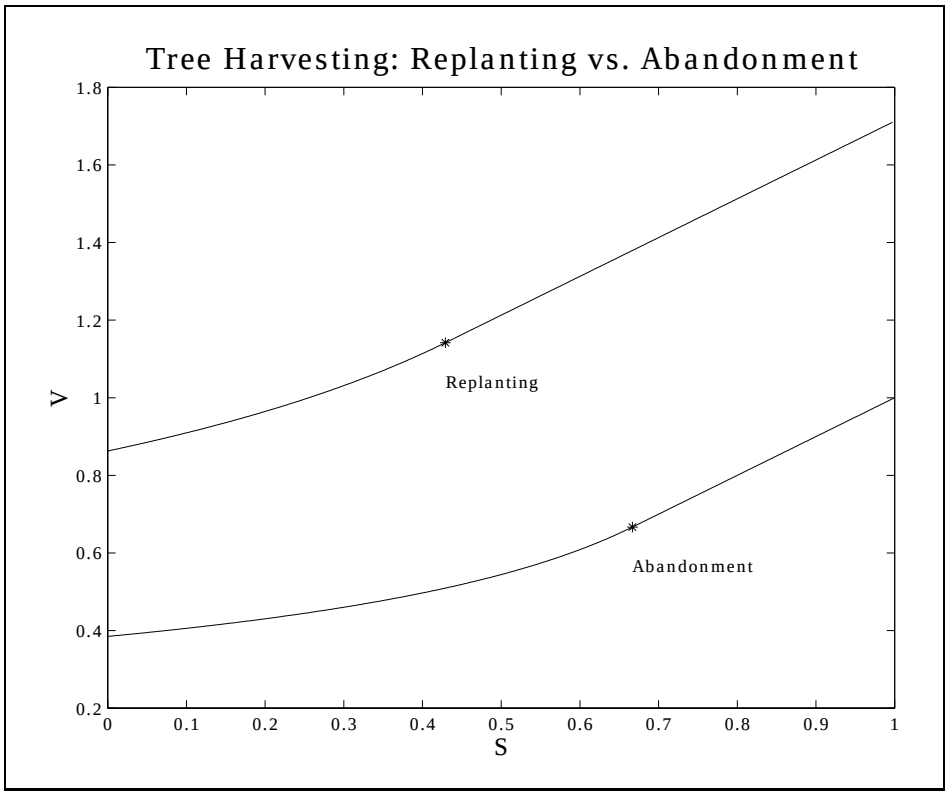

Figure 11.5

The growth function $\alpha(m-S)$ used above was "nice" in allowing a simple solution. It is somewhat limited in its ability to represent biological phenomena. Simple extensions, however, do not necessarily have simple solutions

and we will need to resort to numerical methods. This is also true if randomness is added to the growth process. For example, let us complicate the growth process by defining the timber stand to be governed by

$$dS = \alpha(m - S)(b + S)dt + \sigma(S)dz.$$

It is reasonable to require that $\sigma(0) = 0$, thereby preventing negative stand sizes. An example is the square root process $\sigma(S) = \sigma\sqrt{S}$. Furthermore, with $b > 0$, the instantaneous mean is quadratic with a positive root, m, and a negative root $-b$; if $b < m$ the mean is increasing for $S < (m - b)/2$, whereas if $b > m$ the mean is decreasing for all positive S. The Bellman's equation is

$$\rho V = \alpha(m - S)(b + S)V'(S) + \tfrac{1}{2}\sigma^2 SV''(S),$$

with the same boundary conditions as in the deterministic problem.[17] We defer discussion of numerical solution of this problem to the next chapter (Section ??).

11.2.2 Barrier Control

In barrier control problems it is optimal to maintain the state within a region by keeping it on the region's boundary whenever it would otherwise tend to move outside of it and to do nothing when the state is in the interior of the region. This, of course, assumes that the state is sufficiently controllable so that such a policy is feasible. Barrier control problems can be thought of as limiting cases of impulse control problems as the size of any fixed costs go to zero. When this happens, the size of the jump goes to zero, so the trigger and target points become equal. This represents something of a dilemma because the value matching condition between the target and jump points becomes meaningless when these points are equal. The resolution of this dilemma is to shift the value matching condition to the first derivative and the smooth pasting to the second derivatives.

Example: Capital Investment
Consider an investment situation in which a firm can add to its capital stock,

[17] It might seem odd that introducing the second order term, V_{SS}, in the stochastic case does not require an additional boundary condition. This is a case, however, in which a singularity exists at the boundary $S = 0$ because $\sigma(0) = 0$ (see discussion in Section 11.1.2).

K, at a cost of c per unit. The capital produces output at rate $q(K)$ and the net return on that output is P. Hence the reward function facing the firm is

$$f(K, P, I) = Pq(K) - cI.$$

K is clearly a controllable state, with

$$dK = Idt.$$

P, on the other hand, is stochastic and is assumed to be governed by

$$dP = \mu P dt + \sigma P dz,$$

(geometric Brownian motion). Using a discount rate of ρ, the Bellman equation for this problem is

$$\rho V(K, P) = Pq(K) - cI + IV_K(K, P) + \mu P V_P(K, P) + \tfrac{1}{2}\sigma^2 P^2 V_{PP}(K, P).$$

There are, however, no constraints on how fast the firm can add capital and hence it is reasonable to suppose that, when it invests, it does so at an infinite rate, thereby keeping its investment costs to a minimum.

The optimal policy, therefore, is to add capital whenever the price is high enough and to do so in such a way that the price remains on or below a curve $P^*(K)$. Below this curve no investment takes place and the value function therefore satisfies

$$\rho V(K, P) = Pq(K) + \mu P V_P(KP) + \tfrac{1}{2}\sigma^2 P^2 V_{PP}(KP).$$

This is a simpler expression because, for a given K, it can be solved more or less directly. It is easily verified that the solution has the form

$$V(K, P) = A_1(K)P^{\beta_1} + A_2(K)P^{\beta_2} + \frac{Pq(K)}{\rho - \mu}$$

where the β_i solves $\frac{1}{2}\sigma^2\beta(\beta - 1) + \mu\beta - \rho = 0$. It can be shown that $\beta_2 < 0 < 1 < \beta_1$. For the assumed process for P, 0 is an absorbing barrier so the term associated with the negative root must be forced to equal zero by setting $A_2(K) = 0$ (we will henceforth drop the subscripts on $A_1(K)$ and β_1).

At the barrier, the marginal value of capital must just equal the investment cost:

$$V_K(K, P) = c. \tag{14}$$

Consider now the situation in which the firm finds itself above the barrier (for whatever reason). The optimal policy is immediately to invest enough to bring the capital stock to the barrier. The value of the firm for states above the barrier, therefore, is equal to the value at the barrier (for the same P) less the cost of the new capital:

$$V(K,P) = V(K^*(P),P) - c(K^*(P) - K)$$

where $K^*(P)$ is the inverse of $P^*(K)$. This suggests that the marginal value of capital when the state is above the barrier equals c and hence does not depend on the current price. Thus, in addition to (14), it must be the case that

$$V_{KP}(K,P) = 0. \tag{15}$$

The barrier conditions (14) and (15) can be solved to show that

$$P^*(K) = \frac{c(\rho - \mu)}{(\beta - 1)q'(K)}$$

and

$$A'(K) = -\left(\frac{\beta - 1}{c}\right)^{\beta - 1}\left(\frac{q'(K)}{\rho - \mu}\right)^{\beta}.$$

Notice that to determine $A(K)$ and therefore to completely determine the value function, we must solve a differential equation. The optimal policy, however, does not depend on knowing V, and, furthermore, we have enough information now to determine the marginal value of capital for any value of the state (K,P). Examples of the optimal trigger price curve are displayed in Figure 11.6 using the parameters

$$\begin{aligned} \mu &= 0 \\ \sigma &= 0.2 \\ \rho &= 0.05 \\ c &= 1 \end{aligned}$$

and two alternative specifications for $q(K)$:

$$\begin{aligned} q(K) &= \ln(K+1) \\ q(K) &= \sqrt{K}. \end{aligned}$$

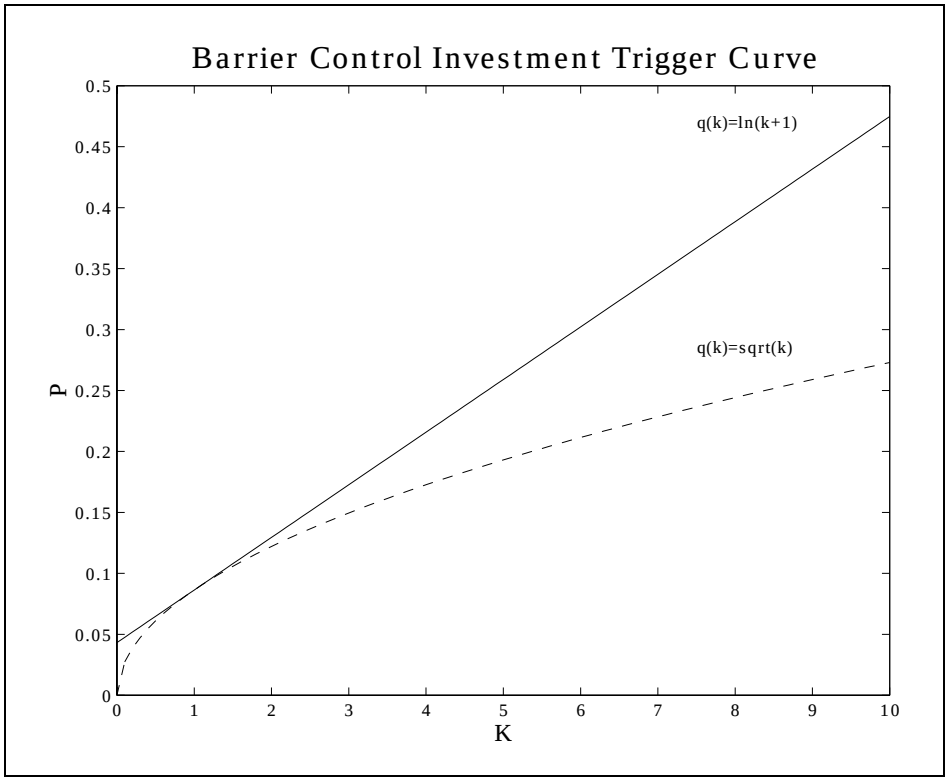

Figure 11.6

11.2.3 Discrete State/Control Problems

We turn now to problems involving transitional boundaries. These can arise because there are discrete states and the optimal time to move from one state to another must be determined. For example, the investment problem described above as an optimal stopping problem can be made more complicated and realistic by allowing disinvestment. The problem then becomes one of determining the value of the project if active, given that one can deactivate it, together with the value of the project if inactive, given that it can be activated. The state here is a stochastic variable that determines the return stream of the activated project. The solution involves two boundaries, one which determines when the project should be activated (given that it is currently inactive), the other when it should be deactivated (given that it is

currently active).[18]

The hallmark of transitional boundary problems is that there is a distinct value function on either side of the boundary and there are conditions that must apply to both of these functions at the boundary. Thus the boundary and the value functions on both sides must all be simultaneously determined. For arbitrary specifications of the boundary, we require that the two value functions are equal at the boundary (value matching) and for the optimal boundary, we require that their derivatives are equal at the boundary (smooth-pasting or high contact).

Example: Entry/Exit

Let us develop in more detail the entry/exit problem mentioned above. A firm can either be not producing at all or be actively producing q units of a good per period at a cost of c per unit . The state variable in this case is the return per unit of output, P, which is a geometric Brownian motion process:

$$P_t = \mu P dt + \sigma P dz.$$

We assume there are fixed costs of activating and deactivating of I and E, with $I + E \geq 0$ (to avoid arbitrage opportunities). The value function is

$$V(P, \delta) = E\left[\int_0^\infty e^{-\rho t}\delta(P - c)dt\right] - \text{ the discounted costs of switching states,}$$

where $\delta = 1$ if active, 0 if inactive.

For positive transition costs, it is reasonable that such switches should be made infrequently. Furthermore it is intuitively reasonable that the optimal control is to activate when P is sufficiently high, $P = P_h$, and to deactivate when the price is sufficiently low, $P = P_l$. It should be clear that $P_l < P_h$, otherwise infinite transactions costs would be incurred. The value function can therefore be thought of as a pair of functions, one for when the firm is active, V^a, and one for when it is inactive, V^i. The former is defined on the interval $[P_l, \infty)$, the latter on the interval $[0, P_h]$. On the interior of these regions the value functions satisfy the Feynman-Kac equations

$$\begin{aligned} \rho V^a &= P - c + \mu P V^a_P + \sigma^2 P^2 V^a_{PP} \\ \rho V^i &= \mu P V^i_P + \sigma^2 P^2 V^i_{PP} \end{aligned} \quad . \tag{16}$$

[18]More accurately, there is an additional, binary, state variable that is 0 if the project is inactive and 1 if it is active. The control is binary, being equal to 0 if the project is active and 1 if it is inactive.

At the upper boundary point, P_h, the firm will change from being inactive to active at a cost of I. Value matching requires that the value functions differ by the switching cost: $V^i(P_h) = V^a(P_h) - I$. Similarly at the point P_l the firm changes from an active state to an inactive one; hence $V^i(P_l) - E = V^a(P_l)$.

Value matching holds for arbitrary choices of P_l and P_h. For the optimal choices the smooth pasting conditions must also be satisfied:

$$V^i_P(P_l) = V^a_P(P_l)$$

and

$$V^i_P(P_h) = V^a_P(P_h).$$

For a geometric Brownian motion process the solution is known for arbitrary levels of P_l and P_h. The general form of the solution is

$$V^a = A^a_1 P^{\beta_1} + A^a_2 P^{\beta_2} + P/(\rho - \mu) - c/\rho$$

$$V^i = A^i_1 P^{\beta_1} + A^i_2 P^{\beta_2}$$

where the four A terms will be pinned down by the boundary conditions and the β solve

$$\tfrac{1}{2}\sigma^2\beta(\beta - 1) + \mu\beta - \rho = 0.$$

It can be shown that, for $\rho > 0$, one of the β is negative and the other is greater than one; define $\beta_1 > 1$ and $\beta_2 < 0$. (It is easy to verify that these solutions solve (16)).

Two of the unknown constants can be eliminated by considering the boundary conditions at $P = 0$ and $P = \infty$. At $P = 0$ only V^i is defined and the geometric Brownian motion process is absorbed; hence $V^i(0) = 0$, which requires that $A^i_2 = 0$. For large P, only V^a is defined and the probability of deactivation becomes vanishingly small; hence the value function would approach $P/(\rho - \mu)$, requiring that $A^a_1 = 0$.

We still have two unknown constants to determine, A^i_1 and A^a_2 (we shall henceforth refer to these as A_1 and A_2, as there is no possible confusion concerning which function they belong to). The value matching conditions require that,

$$V^a(P_h) - I = A_2 P_h^{\beta_2} + P_h/(\rho - \mu) - c/\rho - I = A_1 P_h^{\beta_1} = V^i(P_h)$$

and

$$V^a(P_l) = A_2 P_l^{\beta_2} + P_l/(\rho - \mu) - c/\rho = A_1 P_l^{\beta_1} - E = V^i(P_l) - E.$$

The optimality conditions on P_l and P_h are that the derivatives of V^a and V^i are equal at the two boundary locations:

$$V_P^a(P) = \beta_2 A_2 P^{\beta_2 - 1} + 1/(\rho - \mu) = \beta_1 A_1 P^{\beta_1 - 1} = V_P^i(P)$$

at $P = P_l$ and $P = P_h$. Taken together, the value matching and smooth pasting conditions yield a system of four equations in four unknowns, A_1, A_2, P_l and P_h.

The optimal value functions for a numerical example are illustrated in Figure 11.7. Exogenous parameter values are

μ	σ	ρ	I	E	c
0	0.2	0.05	5	6	1

The endogenous parameters are computed to be

β_1	β_2	A_1	A_2	P_l	P_h
2.1583	-1.1583	3.6299	2.2546	0.4182	2.1996

The optimal boundaries are noted in Figure 11.7 by dashed vertical lines. At P_l, V^i is greater than V^a by the amount E (the cost of switching from active to inactive), whereas, at P_h, V^a exceeds V^i by the amount I (the cost of activating). While less obvious the slopes of the value functions are equal at these two points, as required by the smooth pasting condition. The figure includes extensions of the value functions beyond the ranges for which they are defined. Thus, although, illustrated, V^a for points less than P_l are meaningless in the context of the problem. Similarly, for points above P_h, the function V^i is undefined. We show them to make the smooth pasting conditions easier to see and to point out that the extensions would not satisfy the natural boundary conditions on the problem at $P = 0$ and $P = \infty$.

Another useful way to view the problem is to graph the function $G(P) = V^a(P) - V^i(P)$ (including the extensions of these functions). Several alternative $G(P)$ functions are displayed in Figure 11.8. The one labeled $e = 1$ corresponds to the parameters used to generate Figure 11.7. The smooth pasting conditions ensure that the points $P = P_l$ and $P = P_h$ are stationary points, which occur at the local minimum and maximum values of $G(P)$; the value matching conditions ensure that $G(P_l) = -E$ and $G(P_h) = I$.

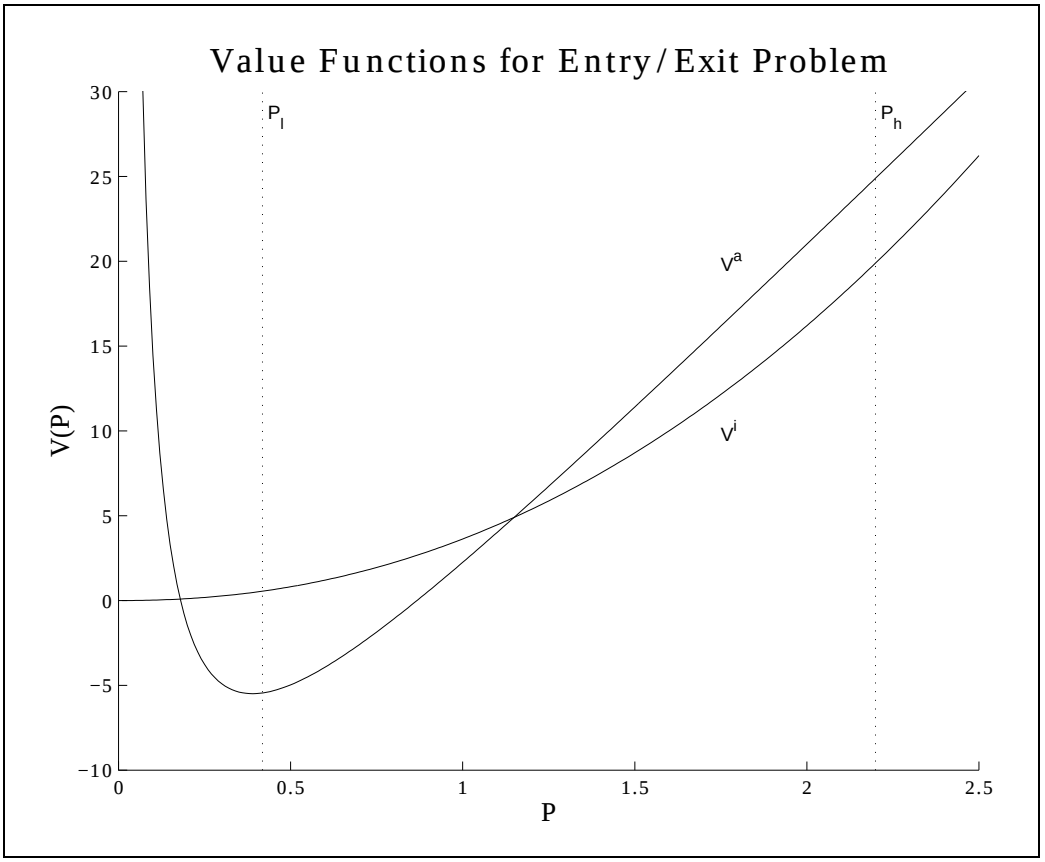

Figure 11.7

Consider now what happens as the switching costs get small, say by multiplying I and E by a smaller and smaller number e. The effect on $G(P)$ is shown in Figure 11.8. As $e \to 0$ the values of P_l and P_H collapse towards each other and the local minimum and maximum values at these points collapse towards zero. In the limit at $e = 0$ it is intuitive that $P_l = P_h = P^*$. Furthermore, at $P = P^*$ the function $G(P)$ must exhibit an inflection point: $G_{PP}(P^*) = 0$. If we impose the three conditions that V^a and V^i match up to their second derivatives at P^*, we can determine the values of A_1, A_2 and P^*. Some tedious algebra will reveal the intuitively reasonable (perhaps obvious) result that $P^* = c$, i.e., the optimal policy is to be active if the current price covers the variable costs (c) and to be inactive otherwise.

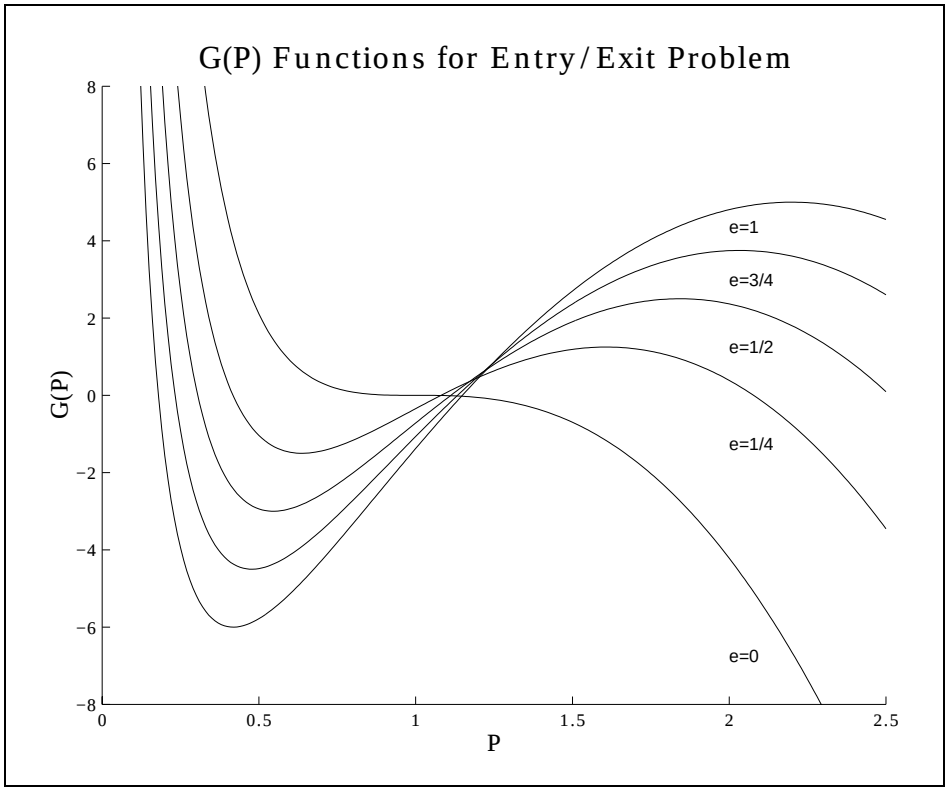

Figure 11.8

Although far from a proof, we hope the intuition behind the limiting process here is clear. The situation is much like passing from an impulse control problem to a barrier control problem as the fixed costs of taking an action go to zero. In that case, as here, the value matching and smooth pasting conditions need to be supplemented with a zero condition on the second derivative. As we shall see in the next section, a similar situation holds in the case of bang-bang control problems.

MATLAB code to solve the entry/exit problem is displayed in Code Box 3. The code is relatively simple, the first section returning values of the four residual equations and the second initializing variables and calling the root finding algorithm. There is one point of interest in the way the code is written. We use the logs of P_l and P_h rather than their levels to prevent negative values from causing the program to act badly (in MATLAB taking

a non-integer power of a negative number results in a complex number which it is better to avoid; the program will continue to run even though it is producing garbage). Specifying the prices in logs does not guarantee that a solution will be found, but it is far more robust to poor starting values than if the levels are used.

Code Box 11.3: Entry/Exit Problem

Optimal Stopping Problems

The optimal stopping problem is in many ways the simplest of the free boundary problems and arises in situations involving a once and for all decision. For example, suppose a firm is attempting to decide whether a certain project should be undertaken. The value of the project depends on a stochastic return that the project, once developed, will generate. The state variable can therefore be taken to be the present value of the developed project. Furthermore, the firm must invest a specified amount to develop the project. In this simple framework, the state space is partitioned into a region in which no investment takes place (when the present value of the developed project is low) and a region in which the project would be undertaken immediately. The boundary between these two areas represents the value of the state, that, if reached from below, would trigger the investment.

It is important to emphasize that optimal stopping problems, although they have a binary control, differ from other binary control problems in that one value of the control pays out an immediate reward, after which no further decisions are made. The one time nature of the control makes the problem quite different from and, actually, easier to solve than problems with binary controls that can be turned on and off.

Stopping problems in continuous time are characterized by a random state governed by

$$dS = \mu(S)dt + \sigma(S)dz,$$

a reward stream $f(S)$ that is paid so long as the process is allowed to continue and a payout function $R(S)$ that is received when the process is stopped (for now we consider only infinite time discounted time autonomous problems; this will be relaxed presently).

Another way to view the stopping problem is as a problem of choosing an optimal time to stop a process. This leads to the following formal statement of the problem

$$V(S) = \max_{t^*(S)} E\left[\int_0^{t^*(S)} e^{-\rho\tau} f(S)d\tau + e^{-\rho t^*(S)} R(S)\right].$$

This value function is described by the differential equation

$$\rho V(S) = f(S) + \mu(S)V_S(S) + \tfrac{1}{2}\sigma^2(S)V_{SS}(S) \tag{17}$$

The optimal control problem consists of finding the boundary between the regions on which the process should be stopped and those on which it should be allowed to continue. For the present, assume that there is a single such switching point, S^*, with $S < S^*$ indicating that the process should be allowed to continue. Thus the differential equation is satisfied on $[\underline{S}, S^*]$, where $\underline{S}$ is a (known) lower bound on the state.

Any specific choice of a control consists of a choice of the stopping point, say S^τ. At this point the value function, to be continuous, must equal the reward

$$V(S^\tau) = R(S^\tau),$$

known as the *value-matching condition*. The optimal choice of S^τ is determined by the *smooth pasting condition*

$$V_S(S^*) = R'(S^*);$$

the optimal choice of S^τ makes the derivative of the value function equal the derivative of the reward function at the boundary between the continuation and stopping regions. Intuitively, the value matching and smooth pasting conditions are indifference relations; at S^* the decision maker is indifferent between continuing and stopping. The value function must, therefore, equal the reward and the marginal value of an additional unit of the state variable must be equal regradless of whether the process is stopped or allowed to continue.

This is the simplest of the optimal stopping problems. We can make them more complex by allowing time to enter the problem either through non-autonomous rewards, state dynamics or stopping payment or by imposing a finite time horizon. In the following example we examine a finite horizon problem.

Example: Exercising an American Put Option
An American put option, if exercised, pays $K-P$, where K is the exercise or strike price and P is the random price of the underlying asset, which evolves according to

$$dP = \mu(P)dt + \sigma(P)dz.$$

The option pays nothing when it is being held, so $f(P) = 0$. Let T denote the option's expiration date, meaning that it must be exercised on or before $t = T$ (if at all).

In general, the option is written on a traded asset so we may use the form of the Bellman's Equation that is discounted at the risk-free rate and with mean function replaced by $rP - \delta_P$ (see Section 10.1.3):

$$rV = V_t + (rP - \delta_P)\, V_P + \tfrac{1}{2}\sigma^2(P)V_{PP}$$

on the continuation region, where δ represents the income flow (dividend, convenience yield, etc.) from the underlying asset. Notice that the constraint that $t \leq T$ means that the value function is a function of time and so V_t must be include in the Bellman's Equation. The solution involves determining the optimal exercise boundary, $P^*(t)$. For puts $P^*(t)$ is a lower bound so the continuation region on which the Bellman's Equation is defined is $[P^*, \infty)$. The boundary conditions for the put option are

$$\begin{array}{ll} V(P,T) = \max(K-P, 0) & \text{(terminal condition)} \\ V(P^*,t) = K - P & \text{(value matching)} \\ V_P(P^*,t) = -1 & \text{(smooth-pasting)} \end{array}$$

and,

$$V(\infty, t) = 0.$$

Example: Machine Abandonment
Consider a situation in which a machine produces an output worth P per unit time, where

$$dP = \mu P dt + \sigma P dz,$$

i.e., that P is a geometric Brownian motion process. The machine has an operating cost of c per unit time. If the machine is shut down, it must

be totally abandoned and thus is lost. Furthermore, at time T, the machine must be abandoned. At issue is the optimal abandonment policy for an agent who maximizes the flow of net returns from the machine discounted at rate ρ.

For the finite time case define τ as equal to the time remaining until the machine must be abandoned, so $\tau = T - t$ and $d\tau = -dt$. The optimal policy can be defined in terms of a function, $P^*(\tau)$; for $P > P^*(\tau)$ it is optimal to keep the machine running, whereas for $P < P^*(\tau)$ it is optimal to abandon it.

The current value of the operating machine satisfies the Bellman's equation

$$\rho V = P - c - V_\tau + \mu P V_P + \tfrac{1}{2}\sigma^2 P^2 V_{PP}.$$

and boundary conditions

$V(P,0) = 0$	terminal condition
$V_P(\infty,\tau) = (1 - e^{-\rho\tau})/(\rho - \mu)$	natural boundary condition
$V(P^*,\tau) = 0$	value matching condition
$V_P(P^*,\tau) = 0$	smooth pasting condition

The first boundary condition states that the machine is worthless when it must be abandoned. The second condition is derived by considering the expected value of a machine that is never abandoned:

$$V(P,\tau) = \left(\frac{P}{\rho - \mu} - \frac{c}{\rho}\right)\left(1 - e^{-\rho\tau}\right)$$

(the derivation of this result is left as an exercise; p. 312). An alternative upper boundary condition is that $V_{PP}(\infty,\tau) = 0$. The remaining two conditions are the value matching and smooth pasting conditions at $P^*(\tau)$.

Consider first the infinite horizon case, which corresponds to the situation that the machine never need be abandoned at any fixed time. It still may be optimal to abandon it if the price is very low, because the odds that the price rises sufficiently fast would not justify taking current losses from operating the machine. Clearly, $P^*(\infty)$ must be less than c and will equal c when there is no uncertainty ($\sigma = 0$).

To determine $P^*(\infty)$ we solve the optimality conditions when V is not a function of τ. We have seen this problem before; its solution is

$$V(P) = A_1 P^{\beta_1} + A_2 P^{\beta_2} + P/(\rho - \mu) - c/\rho,$$

where β solves

$$\tfrac{1}{2}\sigma^2\beta(\beta-1)+\mu\beta-\rho=0.$$

where A_1 and A_2 are constants to be determined by the boundary conditions. For economically meaningful parameter values, one of the β is negative and the other greater than 1. To satisfy the boundary condition as $P \to \infty$, we set $A_1 = 0$, where β_1 is the positive root.

The value matching and smooth pasting conditions are

$$AP^*(\infty)^\beta + P^*(\infty)/(\rho-\mu) - c/\rho = 0$$

and

$$\beta A P^*(\infty)^{\beta-1} + 1/(\rho-\mu) = 0,$$

which are solved by

$$P^*(\infty) = \frac{(\rho-\mu)\beta}{\rho(\beta-1)}c$$

and

$$A = -\frac{P^*(\infty)^{1-\beta}}{(\rho-\mu)\beta}$$

It should be noted that the ability to derive the infinite horizon cutoff price depends on the assumption that P is a geometric Brownian motion process. Also, even in the geometric Brownian motion case, the finite horizon problem does not possess a closed-form solution and hence must be computed numerically. The nature of the problem is demonstrated in Figure 11.9. Notice that $P^*(0) = c$, i.e., that as the date at which the machine must be abandoned is reached, there is no point operating it unless it is currently profitable to do so.

11.2.4 Stochastic Bang-Bang Problems

Bang-bang control problems arise when both the reward function and the state transition dynamics are linear in the control and the control is bounded. In such cases it is optimal to set the control at either its upper or lower bound. The control problem thus becomes one of dividing the state space into a set of points at which the control is at its upper bound and a set at which it is at

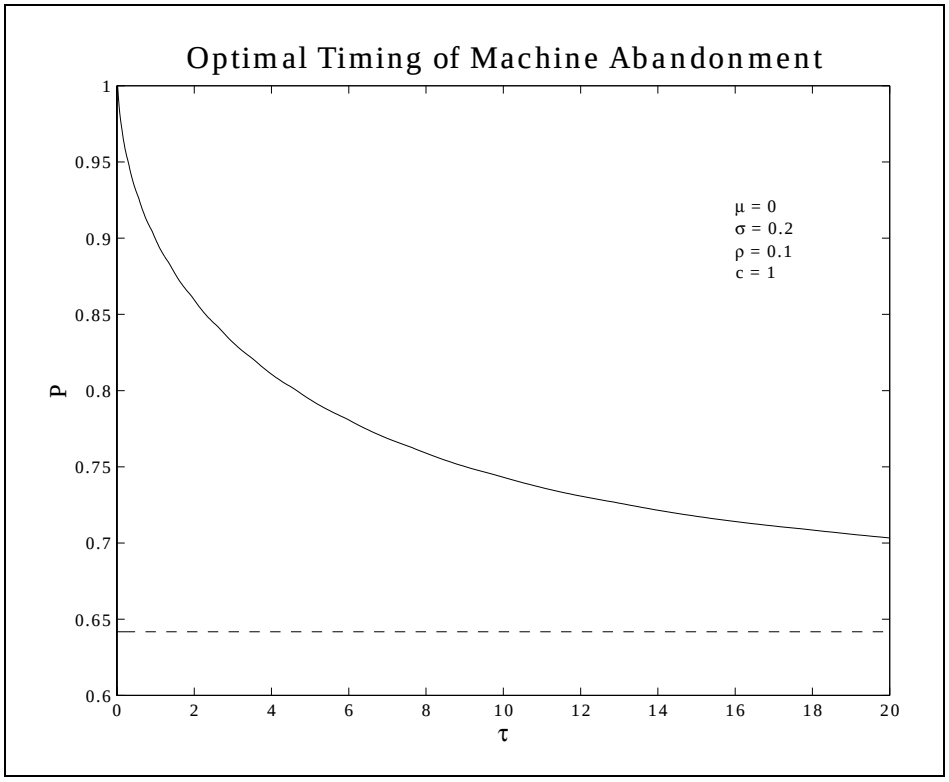

Figure 11.9

its lower bound. Equivalently, the problem is to find the boundary between the two sets. If there is no cost to switching the control from the lower to upper bound, we are in precisely the same situation that we discussed in the last section when the switching costs go to zero. The optimal value function and control is found in a similar fashion: define a Feynman-Kac Equation on each side of the boundary and require that the value functions on either side of the boundary are equal up to their second derivative.

The general bang-bang problem has reward function of the form

$$f_0(S) + f_1(S)x$$

and state dynamics of the form

$$dS = [g_0(S) + g_1(S)x]dt + \sigma(S)dz.$$

Furthermore the control is constrained to lie on a given interval:

$$x_a \leq x \leq x_b.$$

The Bellman's equation for this problem is

$$\rho V = \max_x \; f_0(S) + f_1(S)x + [g_0(S) + g_1(S)x]V_S + \tfrac{1}{2}\sigma^2(S)V_{SS}$$

subject to the control constraint. The Karush-Kuhn-Tucker conditions for this problem indicate that

$$x = \begin{cases} x_a & \text{if } V_S(S) > -\dfrac{f_1(S)}{g_1(S)} \\ x_b & \text{if } V_S(S) < -\dfrac{f_1(S)}{g_1(S)} \end{cases}$$

This suggests that there is a point, S^*, at which

$$f_1(S^*) + g_1(S^*)V_S(S^*) = 0. \tag{18}$$

Assuming that V_S is decreasing in S, this suggests that we must solve for two functions, one for $S < S^*$ that solves

$$\rho V^a = f_0(S) + f_1(S)x_a + [g_0(S) + g_1(S)x_a]V_S^a + \tfrac{1}{2}\sigma^2(S)V_{SS}^a \tag{19}$$

and the other for $S > S^*$ that solves

$$\rho V^b = f_0(S) + f_1(S)x_b + [g_0(S) + g_1(S)x_b]V_S^b + \tfrac{1}{2}\sigma^2(S)V_{SS}^b. \tag{20}$$

We will need three side conditions at S^* to completely specify the problem and to find the optimal location of S^*, namely that

$$\begin{aligned} V^a(S^*) &= V^b(S^*) \\ V_S^a(S^*) &= V_S^b(S^*) \\ V_{SS}^a(S^*) &= V_{SS}^b(S^*). \end{aligned}$$

Combining these conditions with (19) and (20) we see that

$$[f_1(S^*) + g_1(S^*)V_S(S^*)]\, x_a = [f_1(S^*) + g_1(S^*)V_S(S^*)]\, x_b.$$

Clearly this can only be true when the term in the [ ]s is zero, which gives us the optimality result (18).

Example: Harvesting a Renewable Resource

To illustrate the problem consider a manager of a biological (renewable) resource who must determine the optimal harvesting strategy. The state variable, the stock of the resource, is stochastic, fluctuating according to

$$dS = [\alpha S(1-S) - hS]dt + \sigma S dz,$$

where h, the control, is the proportional rate at which the resource is harvested. Assume that the per unit return is p and that $0 \leq h \leq C$. The manager seeks to solve

$$V(S) = \max_h E\left[\int_0^\infty e^{-\rho t} phS dt\right].$$

In the notation of general problem, $x_a = 0$, $x_b = C$, $f_0(S) = 0$, $f_1(S) = pS$, $g_0(S) = \alpha S(1-S)$ and $g_1(S) = -S$. The Bellman equation for this problem is

$$\rho V = \max_h phS + (\alpha S(1-S) - hS) V_S + \tfrac{1}{2}\sigma^2 S^2 V_{SS}.$$

The assumptions the stock dynamics imply that $V(0) = 0$ (once the stock reaches zero it never recovers and hence the resource is worthless). At high levels of the stock, the marginal value of an additional unit to the stock becomes constant and hence $V_{SS}(\infty) = 0$.

The first order conditions for this problem suggest that it is optimal to set $h = C$ if $V_S < p$ and set $h = 0$ if $V_S > p$. The interpretation of these conditions is straightforward: only harvest when the value of a harvested unit of the resource is greater than an unharvested one and then harvest at maximum rate. Thus the problem becomes one of finding the sets

$$S^0 = \{S : V_S > p\}$$

and

$$S^C = \{S : V_S < p\}$$

where

$$\rho V - \alpha S(1-S)V_S - \tfrac{1}{2}\sigma^2 S^2 V_{SS} = 0 \qquad \text{on } S^0$$

and

$$\rho V - (\alpha S(1-S) - CS) V_S - \tfrac{1}{2}\sigma^2 S^2 V_{SS} - pCS = 0 \quad \text{on } S^C$$

The solution must also satisfy the boundary conditions at 0 and ∞ and the continuity conditions at any points S^* such that $V_S(S^*) = p$. The fact that $\alpha S(1-S) - hS$ is concave in S implies that S^* will be a single point, with $S^0 = [0, S^*)$ and $S^C = (S^*, \infty)$.

Figure 11.10 illustrates a numerical approximation to the value function for the problem with $p = C = 1$, $\rho = 0.05$, $\alpha = 0.1$, $\sigma = 0.2$. Figures 11.11 and 11.12 display the first and second derivatives of the value function. Notice that the second derivative has a kink point at S^*. This illustrates the continuity of V and its first two derivatives when S^* is choosen optimally, but also suggests that attempting to approximate the value function with single smooth function will prove problematic. We return to this issue in the next chapter (Section ??).

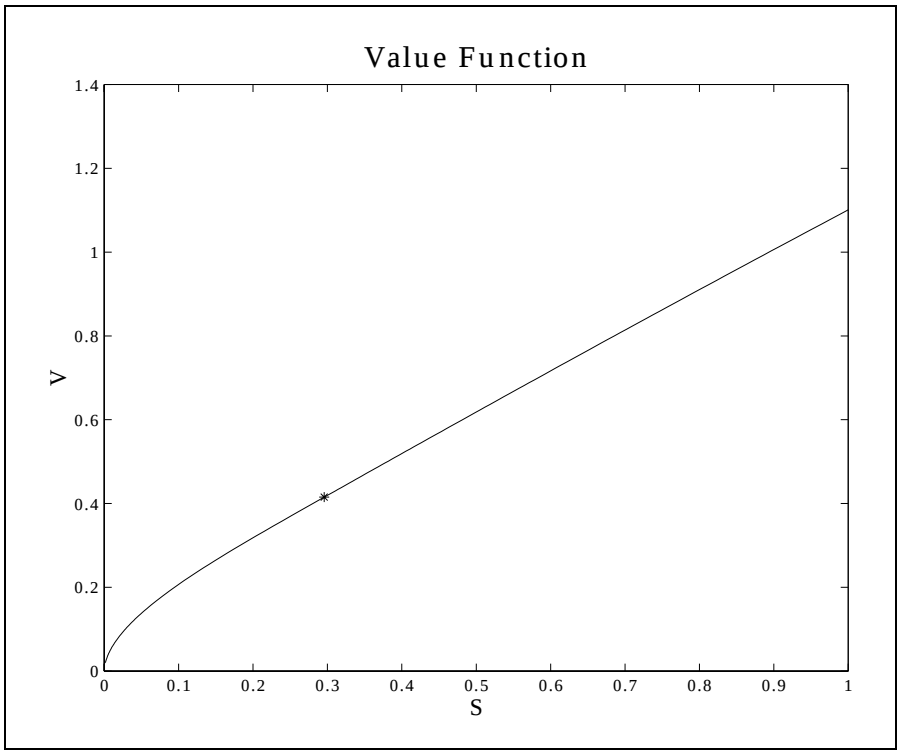

Figure 11.10

The stochastic bang-bang problem generally requires numerical methods to find the optimal switching point. Solutions to deterministic versions of the

Example Box 11.4: Harvesting a Renewable Resource

Problem:

$$\max_h E\left[\int_0^\infty e^{-\rho t} phS dt\right]$$

s.t.

$$dS = [\alpha S(1-S) - hS]dt + \sigma S dz,$$

and

$$0 \leq h \leq C$$

Variables:

h the proportional harvest rate (the control)
S the stock of the resource (the state)

Parameters:

$$\alpha, \sigma, \rho$$

Bellman equation

$$\rho V = \max_h phS + (\alpha S(1-S) - hS)\, V_S + \tfrac{1}{2}\sigma^2 S^2 V_{SS}.$$

Boundary Conditions:

$$V(0) = 0$$
$$V_{SS}(\infty) = 0$$

Optimality Conditions:

$h = C$ if $V_S < p$
$h = 0$ if $V_S > p$

or

$$V_S(S^*) = p$$

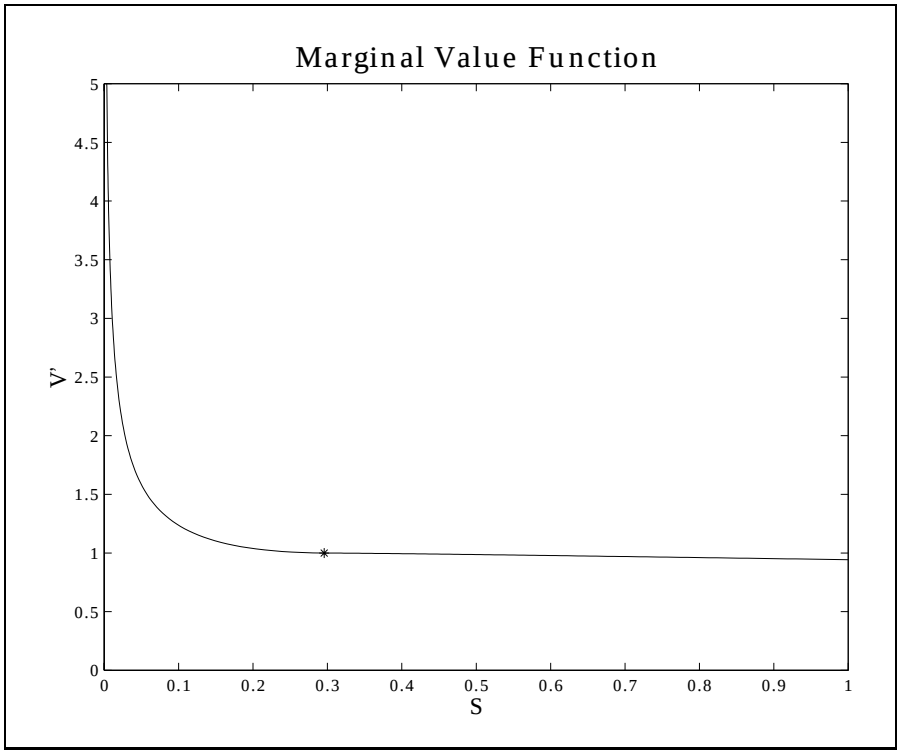

Figure 11.11

problem, however, can often be found directly or by simple application of a root finding algorithm. In the deterministic version, it can be shown that the value matching condition implies the smooth-pasting condition. Also there is no need to specify anything about the second derivative because it drops out of the Bellman's Equation when $\sigma = 0$. The optimal switching point can be shown to satisfy the following condition (the proof is tedious so we've put it in an appendix)

$$\frac{1}{\rho}\frac{d\left(f_0(S) - g_0(S)\dfrac{f_1(S)}{g_1(S)}\right)}{dS} + \frac{f_1(S)}{g_1(S)} = 0.$$

Notice also that the optimal trigger stock does not depend on the capacity constraint levels (x_a and x_b). There is a condition that we have not mentioned, however, that is needed for a well defined solution. We require

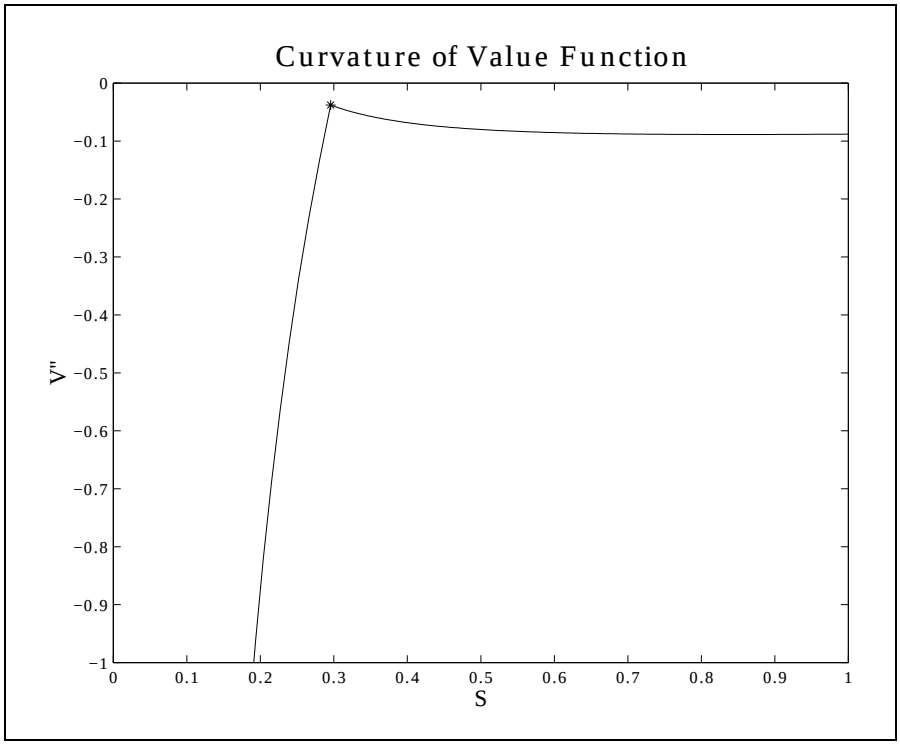

Figure 11.12

that

$$x_a < -g_0(S^*)/g_1(S^*) < x_b.$$

This condition ensures that setting $x = x_a$ when $S < S^*$ will cause the stock level to increase and setting $x = x_b$ when $S > S^*$ will cause the stock to decrease (technically, this says that S is a controllable process).

As an example consider the deterministic version of the optimal harvesting problem. The reward function is

$$f(S, h) = phS$$

and the state is governed by

$$dS = [\alpha S(1 - S) - hS]dt.$$

The optimality condition becomes

$$p\frac{\alpha}{\rho}\frac{dS(1-S)}{dS} - p = 0,$$

which is solved at $S^* = \frac{1}{2}(1-\rho/\alpha)$. Note that the maximum sustainable yield occurs at $S = \frac{1}{2}$, so this rule suggests it is optimal to harvest to the point that the sustained yield is less than maximum (so long as $\rho > 0$). Also notice that when $\alpha < \rho$ we are in a situation where the stock grows so slowly that it is optimal to drive it to extinction.[19]

Consider what happens as we let the capacity constraint, C get large. In the limit, as $C \to \infty$, the problem is transformed onto a barrier control problem in which it is optimal to maintain the stock level at or below a free boundary point P^*. In this case it would only be necessary to determine the value function below the boundary. This would satisfy the Bellman's Equation

$$\rho V = \alpha S(1-S)V_S + \tfrac{1}{2}\sigma^2 S^2 V_{SS}$$

along with the boundary conditions:

$$V_S(S^*) = p$$

and

$$V_{SS}(S^*) = 0.$$

An intuitive way to see that these conditions are correct is to consider the value of an addition unit of the resource when $S \geq S^*$. The additional unit would be harvested immediately and generate a return of p. This means that the value function is linear for $S \geq S^*$ and hence the second derivative of the value function must be zero. Continuity of the first and second derivatives at S^* then gives us the two boundary conditions.

Example: Production with a Learning Curve

More complicated bang-bang problems arise when there are two state variables. The free boundary is then a curve, which typically must be approximated. An example comes from Majd and Pindyck (1989), which develops

[19] To avoid misunderstanding we stress that this simple model only values the resource for the money its harvesting brings; no normative implications about the social value of this rule should be drawn.

a model of production with learning-by-doing. Up to some limit, a firm can reduce it marginal costs the more of a good it produces. It therefore has an incentive to produce more than it otherwise might due to the future cost reductions it thereby achieves. In their model, Majd and Pindyck assume that marginal and average costs are constant at any point in time but decline at an exponential rate in cumulative production until a minimum marginal cost level is achieved. They derive an optimal production rule for a firm maximizing the present value of returns (price less cost times output) over an infinite horizon. The model is summarized in Example Box 5.

The Bellman equation (shown in the Example Box) uses the risk-free discount, r, and the "risk-free" mean, $r-\delta$ (see Section 10.1.3 for discussion). Notice that Bellman's Equation is linear in output (x) and hence the solution is of the bang-bang variety. The optimal control satisfies the KKT conditions:

$$P - C(Q) + V_Q - \lambda \leq 0,\ x \geq 0, \text{ and C.S.}$$

and

$$x_c - x \geq 0,\ \lambda \geq 0, \text{ and C.S.}$$

These conditions are satisfied by choosing x to equal either 0 or x_c according to:

$$x = 0 \text{ if } P + V_Q < C(Q)$$

$$x = x_c \text{ if } P + V_Q > C(Q).$$

Substituting the optimal production rate into the Bellman Equation and rearranging yields

$$rV(P,Q) = (r-\delta)PV_P(P,Q) + \tfrac{1}{2}\sigma^2 P^2 V_{PP} + \max(0, P - C(Q) + V_Q(P,Q))x_c,$$

a partial differential equation.

The boundary conditions for this problem require that

$$V(0,Q) = 0$$

$$V_P(\infty,Q) = x_c/\delta$$

$$V(P,Q_m) = \bar{V}(P) \text{ (defined below)}$$

Example Box 11.5: Production with a Learning Curve

Problem:

$$\max_x \int_0^\infty e^{-\rho t}\,(P - C(Q))\,x\,dt$$

s.t.

$dP = \mu P\,dt + \sigma P\,dz$		price transition equation
$dQ = x\,dt$		cumulative production identity
$C(Q) = \begin{cases} ce^{-\gamma Q} & \text{if } Q < Q_m \\ ce^{-\gamma Q_m} = \bar{c} & \text{if } Q \geq Q_m \end{cases}$		marginal cost function
$0 \leq x \leq x_c$		control constraint

Variables:

P	output price (uncontrolled state)
Q	cumulative production (controlled state)
x	current production rate (control)

Parameters:

μ	rate of expected price appreciation
ρ	discount rate
r	risk free interest rate
δ	rate of return shortfall $(\rho - \mu)$
c	initial marginal cost
$\bar{c}$	minimum marginal cost
Q_m	minimum production associated with minimum cost
x_c	maximum production rate (capacity)

Bellman's Equation:

$$rV = \max_x\,(P - C(Q))\,x + xV_Q + (r-\delta)PV_P + \tfrac{1}{2}\sigma^2 P^2 V_{PP}$$

$$\text{s.t. } 0 \leq x \leq x_c$$

Optimal Control (bang-bang):

$$P - C(Q) + V_Q < 0 \quad \Rightarrow \quad x = 0$$
$$P - C(Q) + V_Q > 0 \quad \Rightarrow \quad x = x_c.$$

and that V, V_P, and V_Q be continuous. The first boundary condition reflects the fact that 0 is an absorbing state for P; hence is P reaches 0, no revenue will ever be generated and hence the firm has no value. The second condition is derived from computing the expected revenue if the firm always produces at maximum capacity, as it would be if the price were to get arbitrarily large (i.e., if the probability that the price falls below marginal cost becomes arbitrarily small). The derivative of the expected revenue is x_c/δ.

The third boundary condition is a "terminal" condition in Q. Once Q_m units have been produced the firm has reached its minimum marginal cost. Further production decisions do not depend on Q nor does the value of the firm, V. An explicit solution can be derived for $Q > Q_m$:

$$\bar{V}(P) = \begin{cases} A_1 P^{\beta_1} & \text{if } P \leq \bar{c} \\ A_2 P^{\beta_2} + \frac{P}{\delta} - \frac{\bar{c}}{r} & \text{if } P \geq \bar{c}, \end{cases}$$

where the β solve the quadratic equation

$$\tfrac{1}{2}\sigma^2\beta(1-\beta) + (r-\delta)\beta - r = 0$$

and the A_1 and A_2 are computed using the continuity of $\bar{V}$ and $\bar{V}_P$.

The continuity requirements on the value function, even though the control is discontinuous, allow us to determine a free boundary between the regions of the state space in which production will and will not occur. Intuitively, there is a function $P^*(Q)$ above which the price is high enough to justify current production and below which no production is justified.

Notice that below the free boundary the Bellman's equation takes a particularly simple form

$$rV(P,Q) = (r-\delta)PV_P(P,Q) + \tfrac{1}{2}\sigma^2 P^2 V_{PP},$$

which together with the first boundary condition ($V(0,Q) = 0$), is solved by

$$V(P,Q) = A_1(Q)P^{\beta_1},$$

where $A_1(Q)$ is yet to be determined. Above the boundary, however, there is no closed form solution. $A_1(Q), P^*(Q)$ and $V(P,Q)$ for $P \geq P^*$ must be computed numerically. Figure 11.13 illustrates the problem using the base parameters in Majd and Pindyck. Solution methods for this problem are presented in the next chapter (Section ??).

The solution methods for this problem depend on being able to determine the position of the free boundary. It is therefore worth exploring some of the

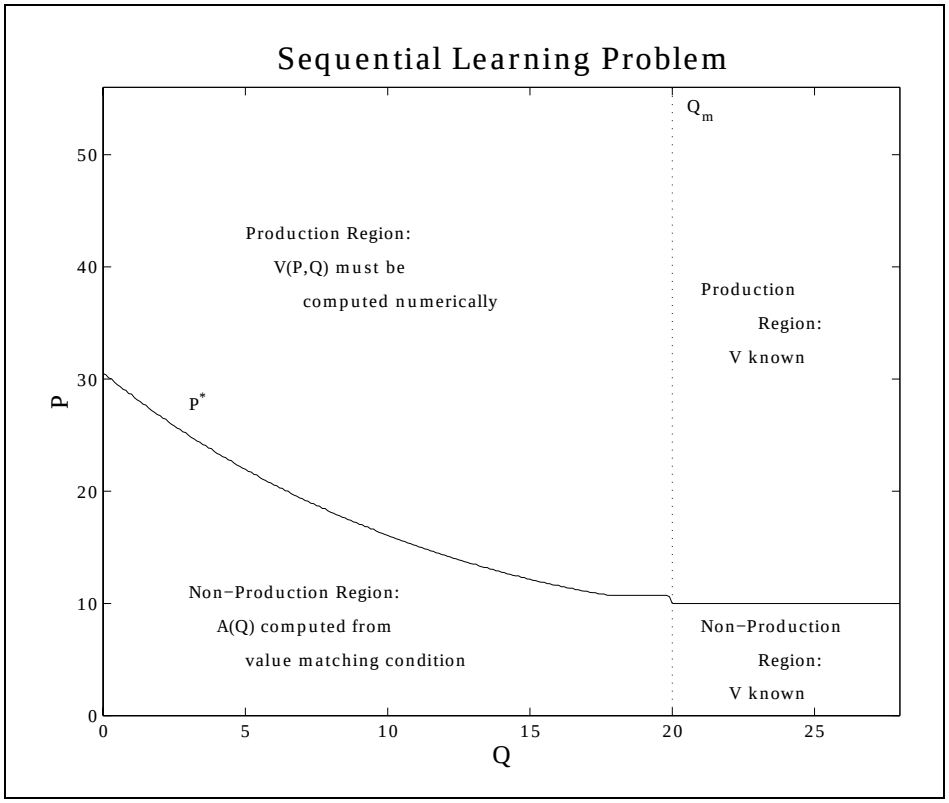

Figure 11.13

consequences of the continuity conditions on V. First, consider the known form of the value function below the free boundary and its derivative:

$$V(P,Q) = A_1(Q)P^{\beta_1}$$

$$V_P(P,Q) = \beta_1 A_1(Q)P^{\beta_1 - 1}.$$

Eliminating $A_1(Q)$ yields

$$PV_P(P,Q) = \beta_1 V(P,Q).$$

This condition holds everywhere below the boundary and at it as well. By the continuity of the V and V_S, it must also hold as the boundary is approached from above.

Another relationship that is useful to note concerns the continuity in the Q direction. Below the boundary,

$$V_Q(P,Q) = A_1'(Q)P^{\beta_1}.$$

The derivative of A_1 is constant in P and may therefore be related to V_Q as it approaches the boundary from above, which is known from the Bellman equation:

$$\begin{aligned} V_Q(P,Q) &= A_1'(Q)P^{\beta_1} \\ &= (\mathcal{L}V(P^*,Q) - (P^* - C(Q))) \left(\frac{P}{P^*}\right)^{\beta_1} \end{aligned}$$

where the differential operator $\mathcal{L}$ is defined as

$$\mathcal{L}V(P,Q) = rV(P,Q) - (r-\delta)PV_P(P,Q) - \tfrac{1}{2}\sigma^2 P^2 V_{PP}(P,Q)$$

But we have already seen that $P^* - C(Q) + V_Q(P^*,Q) = 0$ and therefore $\mathcal{L}V(P^*,Q) = 0$. Summarizing these results, we see that

$$V_Q(P,Q) = \begin{cases} -(P^* - C(Q))\left(\frac{P}{P^*}\right)^{\beta_1} & \text{for } P \le P^* \\ \mathcal{L}V(P,Q) - (P - C(Q)) & \text{for } P \ge P^* \end{cases}$$

It is clear in this expression that V_Q is continuous at P^*.[20]

[20]We should note that our treatment differs somewhat from that of Majd and Pindyck. They discuss only two boundary conditions at $P^*(Q)$, value matching and $P^* - C(Q) + V_Q(P^*,Q) = 0$. To see that this is insufficient, consider the following form for the value function above the free boundary

$$V(P,Q) = A_2 P^{\beta_2} + P/\delta - P^*(Q)/r$$

where

$$P^*(Q) = \frac{r\, e^{\gamma(Q_m - Q)} + \gamma\, e^{-r(Q_m - Q)}}{r+\gamma}\overline{c}$$

This function satisfies the Bellman equation, and the condition that

$$P^* - C(Q) + V_Q(P^*,Q) = 0.$$

Below the boundary the solution has the form $A(Q)P^{\beta_1}$, so $A(Q)$ is a free parameter that is determined by the value matching condition (it can be shown that this is the optimal boundary for the deterministic problem).

11.3 End Notes

11.3.1 Bibliographic Notes

The renewable resource harvesting problem is from Pindyck, the optimal investement from Cox, Ingersoll and Ross, the portfolio choice example from Merton.

Free boundary problems are increasingly common in economics. Dixit (1991), Dixit (1993) and Dixit and Pindyck contain useful discussions of these problems. Several of the examples are discussed in these sources.

The original solution to the timber harvesting problem with replanting is attributed to Martin Faustmann, who discussed it in an article published in 1849. Irving Fisher discussed the related problem with abandonment in The Theory of Interest. For further discussion see Gaffney (1960), Hershleifer (1970). To our knowledge, the problem has never been discussed in print as a stochastic continuous time problem.

The entry/exit example originates with Brennan and Schwartz and McDonald and Seigel.

Numerous authors have discussed renewable resource management problems. The bang-bang formulation is discussed most fully in a series of papers by Ludwig, where detailed proofs can be obtained. The proof in the appendix to this chapter is modeled after a similar proof in Ludwig (19??).

The cow replacement examples originate with the authors.

11.3.2 References

Faustmann, Martin, "On the Determination of the Value Which Forestland and Immature Stands Possess for Forestry." Trans. M. Gane. Oxford Institute Paper 42 (1968).

Fisher, Irving. The Theory of Interest.

Gaffney, M. Concepts of Financial Maturity of Timber and Other Assets. A.E. Information Series #62, Department of Agricultural Economics, North Carolina State College, 1960.

Hershleifer, J. Investment, Interest and Capital. Englewood Cliffs, NJ, 1970.

Merton, Robert C. "An Asymptotic Theory of Growth Under Uncertainty." Review of Economic Studies. 42(1975):375-393.

[INCOMPLETE]

Appendix A: Deriving the Boundary Conditions for Resetting Problems

It is instructive to view the resetting problem from another perspective. In a simple resetting problem an asset is replaced at a discrete set of times when $S = S^*$, at which point a reward, $f(S^*)$ is obtained. Let us define $\tau(S, S^*)$ to be the (random) time until the state first hits S^*, given that it is now equal to S. The first time the state hits S^* a reward worth $f(S^*)e^{-\rho\tau(S,S^*)}$ (in current units of account) will be generated and the state is reset to 0. The time elapsing after a resetting until the state next hits S^* depends on a random variable that has the same distributional properties as $\tau(0, S^*)$ and is independent of previous hitting times (by the Markov property). The expected discounted rewards (i.e., the value function) can be therefore be written as

$$\begin{aligned} V(S;S^*) &= f(S^*)E\left[e^{-\rho\tau(S,S^*)}\right]\sum_{i=0}^{\infty}\left(E\left[e^{-\rho\tau(0,S^*)}\right]\right)^i \\ &= \frac{f(S^*)E\left[e^{-\rho\tau(S,S^*)}\right]}{1-E\left[e^{-\rho\tau(0,S^*)}\right]}. \end{aligned}$$

To simplify the notation, let

$$\beta(S,S^*) = E\left[e^{-\rho\tau(S,S^*)}\right],$$

so the value function is

$$V(S;S^*) = \frac{f(S^*)\beta(S,S^*)}{1-\beta(0,S^*)}.$$

From the definition of τ it is clear that $\tau(S^*,S^*)=0$ so $\beta(S^*,S^*)=1$. Hence the boundary condition that

$$V(S^*;S^*) = \frac{f(S^*)}{1-\beta(0,S^*)}$$

Combining this with the lower boundary condition

$$V(0;S^*) = \frac{f(S^*)\beta(0,S^*)}{1-\beta(0,S^*)}$$

leads to the value matching condition that

$$V(S^*;S^*) = V(0;S^*) + f(S^*).$$

Notice that value matching does not indicate anything about the optimality of the choice of S^*. One way to obtain an optimality condition is to set the derivative of $V(S,S^*)$ with respect to S^* equal to zero. After suitable rearrangement the FOC is, for evert S,

$$f'(S^*)\beta(S,S^*) + f(S^*)\left[\frac{\partial\beta(S,S^*)}{\partial S^*} + \frac{\beta(S,S^*)}{1-\beta(0,S^*)}\frac{\partial\beta(0,S^*)}{\partial S^*}\right] = 0. \quad (21)$$

In order to show that this is equivalent to the smooth pasting condition we will use two properties of β. First, $\beta(S^*,S^*)$ is identically equal to 1, so

$$\left.\frac{dS}{dS^*}\right|_{S=S^*} = 1.$$

This implies that

$$\frac{d\beta(S^*,S^*)}{dS^*} = \frac{\partial\beta(S^*,S^*)}{\partial S} + \frac{\partial\beta(S^*,S^*)}{\partial S^*} = 0$$

and hence that

$$\frac{\partial\beta(S^*,S^*)}{\partial S} = -\frac{\partial\beta(S^*,S^*)}{\partial S^*}.$$

The second fact, a result of the Markov assumption, is that

$$\beta(S,S^*+dS^*) = \beta(S,S^*)\beta(S^*,S^*+dS^*).$$

taking limits as $dS^* \to 0$ we see that

$$\frac{\partial \beta(S, S^*)}{\partial S^*} = \beta(S, S^*) \frac{\partial \beta(S^*, S^*)}{\partial S^*}.$$

If we evaluate (21) at $S = S^*$ and rearrange, it is straightforward to see that

$$\begin{aligned} f'(S^*) &= -f(S^*) \left[\frac{\partial \beta(S^*, S^*)}{\partial S^*} + \frac{\beta(S^*, S^*)}{1 - \beta(0, S^*)} \frac{\partial \beta(0, S^*)}{\partial S^*} \right] \\ &= -f(S^*) \left[1 + \frac{\beta(0, S^*)}{1 - \beta(0, S^*)} \right] \frac{\partial \beta(S^*, S^*)}{\partial S^*} \\ &= -\frac{f(S^*)}{1 - \beta(0, S^*)} \frac{\partial \beta(S^*, S^*)}{\partial S^*} \\ &= \frac{f(S^*)}{1 - \beta(0, S^*)} \frac{\partial \beta(S^*, S^*)}{\partial S} \\ &= \frac{\partial V(S^*, S^*)}{\partial S} \end{aligned}$$

which is the desired result.

Appendix B: Deterministic Bang-Bang Problems

The general form for a deterministic bang-bang type problem has a reward function

$$f_0(S) + f_1(S)x$$

state dynamics

$$dS = [g_0(S) + g_1(S)x]dt$$

and control constraint

$$x_a \leq x \leq x_b.$$

Suppose we use a control, not necessarily optimal, with S^τ as a switching point, e.g., set $x = x_a$ for $S < S^\tau$ and $x = x_b$ for $S > S^\tau$.[21] At $S = S^\tau$ we

[21] This assumes that the state is growing when x_a is used and is shrinking when x_b is used. It is a simple matter to reverse these inequalities.

choose x in such a way that $dS/dt = 0$. Summarizing, define

$$x(S, S^\tau) = \begin{cases} x_a & \text{if } S < S^\tau \\ -\dfrac{g_0(S)}{g_1(S)} & \text{if } S = S^\tau \\ x_b & \text{if } S > S^\tau \end{cases},$$

with $x_a < -g_0(S^\tau)/g_1(S^\tau) < x_b$. The value function satisfies the differential equation

$$V(S, S^\tau) = \frac{1}{\rho}\Big(f_0(S)+f_1(S)x(S, S^\tau)+\Big[g_0(S)+g_1(S)x(S, S^\tau)\Big]V_S(S, S^\tau)\Big), \quad (22)$$

which, evaluated at $S = S^\tau$, yields

$$V(S^\tau, S^\tau) = \frac{1}{\rho}\left(f_0(S^\tau) - f_1(S^\tau)\frac{g_0(S^\tau)}{g_1(S^\tau)}\right). \quad (23)$$

In spite of the discontinuity of the control at S^τ, the value function is continuous, as is readily apparent by writing it as

$$V(S, S^\tau) = \int_0^\infty e^{-\rho t}\left(f_0(S) + f_1(S)x(S, S^\tau)\right) dt,$$

and noting that as S approaches S^τ from below (above), the amount of time during which the control is set at x_a (x_b) goes to 0.

The continuity of V can be used to demonstrate the continuity of $V_S(S, S^\tau)$ at $S = S^\tau$, and to thereby determine its value:[22]

$$V_S(S^\tau, S^\tau) = -\frac{f_1(S^\tau)}{g_1(S^\tau)}. \quad (24)$$

[22]To determine the limit from below, note that continuity of V implies that

$$\begin{aligned} \lim_{S\nearrow S^\tau} V(S, S^\tau) &= \lim_{S\nearrow S^\tau} \frac{1}{\rho}\left[f_0(S) + f_1(S)x_a + \left(g_0(S) + g_1(S)x_a\right)V_S(S, S^\tau)\right] \\ &= \frac{1}{\rho}\left[f_0(S^\tau) + f_1(S^\tau)x_a + \left(g_0(S^\tau) + g_1(S^\tau)x_a\right)\lim_{S\nearrow S^\tau} V_S(S^\tau, S^\tau)\right] \\ &= \frac{1}{\rho}\left[f_0(S^\tau) - \frac{f_1(S^\tau)g_0(S^\tau)}{g_1(S^\tau)}\right] \equiv V(S^\tau, S^\tau). \end{aligned}$$

Rearranging, we see this expression implies that

$$\begin{aligned} \left(g_0(S) + g_1(S)x_a\right)\lim_{S\nearrow S^\tau} V(S, S^\tau) &= -\left(f_1(S)x_a + \frac{f_1(S^\tau)g_0(S^\tau)}{g_1(S^\tau)}\right) \\ &= -\frac{f_1(S^\tau)}{g_1(S^\tau)}\left(g_0(S^\tau) + g_1(S^\tau)x_a\right) \end{aligned}$$

The same exercise can be applied to solving for the limit from below.

So far, however, we have only considered the value function for the control S^τ. To choose the control optimally, we must pick S^τ to satisfy

$$V_{S^\tau}(S, S^\tau) = 0.$$

For $S \neq S^\tau$ we can differentiate (22) to see that

$$V_{S^\tau}(S, S^\tau) = \frac{1}{\rho}\Big[f_1(S)+g_1(S)V_S(S, S^\tau)\Big]x_{S^\tau}(S, S^\tau)+g_1(S)x(S, S^\tau)V_{SS^\tau}(S, S^\tau). \quad (25)$$

However, except at $S = S^\tau$, $x_{S^\tau}(S, S^\tau)$ and $V_{SS^\tau}(S, S^\tau)$ are zero and hence we only need to set this derivative to zero at $S = S^\tau$. (25) is not well defined at $S = S^\tau$ because the derivative $x_{S^\tau}(S, S^\tau)$ is undefined at this point. Instead we use the relationship

$$\frac{dV(S^\tau, S^\tau)}{dS^\tau} = V_S(S^\tau, S^\tau) + V_{S^\tau}(S^\tau, S^\tau).$$

Rearranging this and using (23) and (24) we get

$$\begin{aligned} V_{S^\tau}(S^\tau, S^\tau) &= \frac{dV(S^\tau, S^\tau)}{dS^\tau} - V_S(S^\tau, S^\tau) \\ &= \frac{1}{\rho}\frac{d\left(f_0(S^\tau) - g_0(S^\tau)\frac{f_1(S^\tau)}{g_1(S^\tau)}\right)}{dS^\tau} + \frac{f_1(S^\tau)}{g_1(S^\tau)} \end{aligned}$$

Thus the optimal switching points are found by solving for the roots of this expression.

Ludwig (1979) discusses a case in which there are multiple roots, leading to a situation in which V_S may be discontinuous at a root; this root represents an unstable equilibrium at which x is undefined.

Exercises

1. Optimal Sales from an Inventory
   What follows is an example of a problem with a continuous control that is bounded below by zero. It is not explicitly bounded above but a constraint on the state leads to an optimal control of the bang-bang form (bang-bang problems are discussed further on page 364). Because of the nature of the state transition and the state constraint, the control problem is essentially identical to an optimal stopping problem. Specifically, a firm with an irreplaceable inventory finds it optimal to

either hold the inventory or sell it completely all at once. The sell decision is triggered by the price of the good; if the price is high enough it is optimal to sell. The main problem arises in determining the trigger price.

Consider a situation in which an agent has an inventory of S_0 units of a good in inventory, all of which must be sold within T periods. It costs k dollars per unit in inventory per period to store the good. In this problem there is a single control, the sales rate q, and two state variables, the price P and the inventory level S. The price is an exogenously given Ito process:

$$dP = \mu(P,t)dt + \sigma(P,t)dz.$$

The amount in storage evolves according to

$$dS = -qdt.$$

Furthermore the control must be nonnegative (i.e., the agent cannot purchase additional amounts)

$$q \geq 0$$

and the inventory level must be non-negative:

$$S \geq 0.$$

The problem can be written as

$$V(S,P,t) = \max_{q(S,P,t)} E_t \int_t^T e^{-rt}\left(qP - kS\right) dt$$

subject to the above constraints.[23]

Bellman's equation for this problem is

$$\rho V = \max_q \left(qP - kS\right) + V_t + V_P\mu + \tfrac{1}{2}V_{PP}\sigma^2 - V_S q,$$

[23]In addition to the usual regularity conditions this problem requires a constraint on the expected growth rate of price (if it grows too fast it always pays to hold). Also $E_t[P_{t+\Delta t}|P_t]$ must be an increasing function of P_t.

subject to the non-negativity constraints on S and q.

The constraint on the state variable S is problematic; it clearly constrains the control but it does so in following way:

$$q \leq 0 \text{ if } S = 0.$$

Given that $q \geq 0$, this implies that

$$q = 0 \text{ if } S = 0.$$

To impose this constraint we introduce a multiplier, λ that exhibits complementary slackness with S. The optimality conditions can then be written

$$\begin{aligned} P - V_S - \lambda \geq 0, \quad q \geq 0, \quad \text{C.S.} \\ S \geq 0, \quad \lambda \geq 0, \quad \text{C.S.} \end{aligned}$$

There are three possible solutions to this problem:

$$\begin{aligned} P < V_S &\Rightarrow q = 0 \\ P = V_S &\Rightarrow \text{ indifference} \\ P > V_S &\Rightarrow q = \infty \text{ if } S > 0 \end{aligned}$$

Thus the optimal control is to either sell all of the inventory (if $P > V_S$) or to sell nothing (if $P < V_S$). Only when $P = V_S$ is the agent indifferent between holding and selling (in which case it is harmless to assume that the inventory would be sold).

There is, therefore, a region in which it is optimal to hold onto inventory bounded by a curve, $P^*(t)$, at which it is optimal to sell the whole inventory. Within the holding region the value function satisfies the PDE

$$rV = -k + V_t + \mu(P,t)V_P + \tfrac{1}{2}\sigma^2(P,t)V_{PP}.$$

Furthermore, it is easy to see that the value function is proportional to the level of inventory; hence it is harmless to normalize by setting

$S_0 = 1$. The value function is then a function of the price level and time alone.

At the boundary the value matching and smooth-pasting conditions hold. Consider that, when the inventory is sold it worth P per unit. Hence the value-matching and smooth-pasting conditions are

$$V(P^*(t), t) = P^*(t)$$

and

$$V_P(P^*(t), t) = 1.$$

If the inventory must be sold on or before a fixed date, T, an additional terminal boundary condition requires that

$$V(P, T) = P.$$

There may, in addition, be a lower boundary constraint, for example at $P = 0$. If zero is an absorbing barrier for P then $V(0) = 0$.

2. Show that a utility function of the form $U(C) = (C^{1-\gamma} - 1)/(1 - \gamma)$ implies an optimal consumption rule of the form $C(W) = aW$. Determine the constant a and, in the process, determine the value function and the optimal investment rule $\alpha(W)$.

3. Suppose that there are only two assets available to investors, which are governed by

$$dR = rRdt$$

and

$$dS = \mu Sdt + \sigma Sdz,$$

i.e., R is a risk-free and S a risky asset. The controls for the investors problem are C, the consumption rate, and α, the fraction of wealth held in the risky asset. Write the Bellman's Equation associated with this problem and derive expressions for the optimal controls.

4. Expand the analysis of the resetting problem that begins on page 379 to include the case in which a flow of payments $f(S)$ is received. In particular, show that value matching holds for any choice of the resetting state and that smooth pasting holds for the optimal choice.

5. In the general optimal stopping problem the value function can be written as

$$V(S;S^*) = E\left[\int_0^{\tau(S,S^*)} e^{-\rho t} f(S_t)dt\right] + E\left[e^{-\rho\tau(S,S^*)}\right] R(S^*),$$

where $\tau(S,S^*)$ is the first time the state equals S^* given that it equals S at time 0. Show that value matching holds for arbitrary S^* and the smooth pasting holds for the optimal S^*.

6. Verify that the optimal harvest stand size in the timber management problem given on page 350 is correct.

7. Consider the manager of a cash account subject to random deposits and withdrawals. In the absense of active management the account is described by absolute Brownian motion

$$dS = \mu dt + \sigma dz.$$

The manager must maintain a positive cash balance. When the account hits 0, the manager must draw funds from an interest bearing account. To increase the cash account by z units, the manager bears a cost of $f + cz$, i.e., there are both fixed and proportional variable costs of control. Similarly, the manager can place funds in the interest bearing account by withdrawing an amount z from the cash account, incurring costs of $F + Cz$.

Suppose the manager uses a discount rate of ρ and the interest bearing account generates interest at rate r. It is clear that the manager will want to adjust the account only at discrete times so as to minimize the adjustment costs. A control policy can therefore be described as a choice of three cash levels, $S_1 \leq S_2 \leq S_3$, where S_1 is the amount of the addition to the fund when it hits 0, S_3 is the trigger level for withdrawing funds (adding them to the interest bearing account) and

S_2 is the target level (i.e., $S_3 - S_2$ units are withdrawn when the fund hits S_3).

The value function associated with this problem solves the Bellman equation[24]

$$\rho V(S) = \mu V'(S) + \tfrac{1}{2}\sigma^2 V''(S), \text{ for } S \in [0, S_3]$$

with the side conditions that

$$V(0) = V(S_1) - f - (r/\rho + c)S_1$$

and

$$V(S_3) = V(S_2) - F + (r/\rho - C)(S_3 - S_2).$$

Furthermore, an optimal policy satisfies

$$V'(S_1) = (r/\rho + c)$$

and

$$V'(S_3) = V'(S_2) = (r/\rho - C).$$

The Bellman equation can be solved explicitly:

$$V(S) = A\exp(\alpha S) + B\exp(\beta S),$$

where α and β are choosen to solve the differential equation and A and B are chosen to satisfy the side conditions.

[24]Although it is not necessary to solve the problem, it is useful to understand why these conditions are appropriate. The value function here is interpreted as the present value of the current cash position, which does not depend on how much money is in the interest bearing account at the present moment. Cash pays no current flows and hence the Bellman equation is homogeneous (no reward term). The cost of withdrawing funds from the interest bearing account equals the control cost plus the opportunity cost of the lost interest, which is equal to r/ρ times the amount withdrawn. The cost of adding funds to the interest bearing account equals the control cost less the present value of the interest earned on the funds put into the account (r/ρ times the amount of these funds).

Write a MATLAB procedure that accepts the parameters μ, σ, ρ, r, f, F, c, and C and returns the parameters A, B, α, β, S_1, S_2, and S_3. Also determine how the program needs to be modified if the proportional costs (c and C) are zero. Check the answers you obtain using the following parameter values: $\mu = 0$, $\sigma = 0.5$, $\rho = 0.4$, $r = 0.5$, $f = 1$, $F = 0.5$, $c = 0.1$, and $C = 0.1$. You should obtain the result that $S_1 = 0.7408$, $S_2 = 0.8442$, and $S_3 = 2.2216$.

8. Consider an extension to the renewable resource problem discussed on page 367. Suppose that the harvest rate is still constrained to lie on $[0, C]$ but that it cannot be adjusted instantaneously. Instead assume that the rate of adjustment in the harvest rate, x, must lie on $[a, b]$, with $a < 0 < b$, with the proviso that $x \geq 0$ is $h = 0$ and $x \leq 0$ is $h = C$.

   This problem can be addressed by defining h to be a second state variable with a deterministic state transition equation:

   $$dh = xdt.$$

   The optimal control for this problem is defined by two regions, one in which $x = a$ and one in which $x = b$. The boundary between these regions is a curve in the space $[0, \infty) \times [0, C]$.

   Write the PDEs that must be satisfied by the value functions in each region and the value-matching and smooth pasting conditions that must hold at the boundaries.

9. Consider the optimal management of a renewable resource. Suppose that the stock of the resource evolves according to

   $$dS = \alpha(m - S)Sdt + \sigma Sdz.$$

   The (inverse) demand for the resource is given by

   $$p = D^{-1}(q) = a - bq.$$

   and the cost of harvesting the resource is

   $$\frac{c}{2}\frac{q^2}{S}.$$

Assume the appropriate discount rate is ρ and that the social preference function is consumer surplus less harvesting cost.

(a) Define the social planner's reward function.

(b) Write the Bellman's equation for this problem.

(c) Solve the first order conditions and substitute out the optimal control from Bellman's equation to arrive at a concentrated Bellman's equation.

(d) Discuss a computational strategy to solve for the optimal value function (implement it for extra credit).

Chapter 12

Continuous Time Dynamic Models: Methods

In the previous two chapters we saw how continuous time economic models, whether deterministic or stochastic, result in either ordinary or partial differential equations that must be evaluated subject to some boundary conditions. Ordinary differential equations (ODEs) arise in infinite horizon single state models or in deterministic problems solved in terms of time paths. Partial differential equations (PDEs) arise in models with multiple state variables or in finite horizon control problems. From a numerical point of view the distinction between ODEs and PDEs is less important than the distinction between problems which can be solved in a recursive or evolutionary fashion or those that require the entire solution be computed simultaneously because the solution at one point (in time and/or space) depends on the solution everywhere else.

This is the distinction between initial value problems (IVPs) and boundary value problems (BVPs) that we discussed in Chapter ??. With an IVP, the solution is known at some point or points and the solution near these points can then be (approximately) determined. This, in turn, allows the solution at still other point to be approximated and so forth. When it is possible, it is usually faster to use recursive solution techniques, which include Euler and Runge-Kutta methods for ordinary differential equations and recursive finite difference methods or the method of lines for partial differential equations.

We begin this chapter with a discussion of various approaches to solving PDEs. First we discuss finite difference approaches, which are very widely

used and easy to program. We then discuss the method of lines, which uses finite differences for the state variables and expresses the PDE as a system of ordinary differential equations in time. The method of lines can be extended easily using function approximation methods to represent the solution as a function of the states and an ODE in time. Finally, we discuss using collocation as a general scheme to solve PDEs. The various methods are then applied to the solution of stochastic control problems, including problems involving free boundaries.

There are a number of methods for solving PDEs and stochastic control problems that we do not discuss here. These include binary and trinomial tree methods and simulation methods for solving PDEs and discretizing the state and action space in control problems and solving the related discrete problem. Our main rationale for our choices of what to include is that the methods discussed build on general methods developed in previous chapters. Much of what is discussed here should look and feel familiar to readers that have persevered up to this point. We do, however, include some references to other approaches in the bibliographical notes at the end of the chapter.

12.1 Partial Differential Equations

In the previous two chapters we discussed a number of examples of partial differential equations, like the Black-Scholes option pricing formula, for which there are relatively simple solutions. For most interesting problems, we are not so lucky, however. There are several numerical approaches that can be used, including Monte Carlo simulation, binomial trees, finite difference methods and weighted residual methods.

The only difference between an ordinary differential equation (ODE) and a partial differential equation (PDE) is that the solution to the former is a function of a single variable, whereas the solution to the latter is a function of multiple variables. From a computational point of view, this generally means that PDEs are harder to solve than ODEs and, in fact, if there are too many arguments, it may be nearly impossible to solve without getting very clever.

As with ODEs, the distinction between initial and boundary value problems is relevant for PDEs. For example, a function, $V(S,t)$, that solves a PDE of the form

$$V_t = \rho V + \tfrac{1}{2}\sigma^2 V_{SS},$$

with $V(S,0)$ a known function of S, can be propagated with respect to t, starting at $t = 0$. This approach cannot be used to approximate a function $V(S,R)$ satisfying

$$\rho V = f(S,R) + \tfrac{1}{2}[\sigma_{SS}V_{SS} + 2\sigma_{SR}V_{SR} + \sigma_{RR}V_{RR}]$$

subject to boundary conditions at $S = a_S$, $S = b_S$, $R = a_R$ and $R = b_R$. Instead, this type of problem must be solved simultaneously for all relevant values of (S,R).

12.1.1 Finite Difference Methods for PDEs

For evolutionary PDEs, the most common approach uses finite difference methods, which are relatively easy to understand and implement from scratch, at least for low dimensional problems. Furthermore, they can have good numerical properties, especially if you are not looking for a high degree of accuracy. Essentially finite difference methods amount to replacing terms involving derivatives with difference approximations to those derivatives.

Evolutionary PDEs (often called parabolic PDEs) are characterized by having no second derivatives for one of the variables entering the PDE. The parabolic case is particularly important in economic applications, where the form

$$\rho(S)V(S,t) = f(S) + V_t(S,t) + \mu(S)V_S(S,t) + \tfrac{1}{2}\sigma^2(S)V_{SS}(S,t)$$

is often encountered. We denote the first order variable as t to suggest time because in many applications it will have that interpretation; the essential features, however, is that it is first order in one variable and the function is known at some value of that variable.

In simple applications, the PDE is defined on $[a,b] \times [0,T]$. Boundary conditions are specified at $S = a$, $S = b$ and either at $t = 0$ or $t = T$. An initial value problem with the boundary condition specified at $t = 0$ can be propagated forward in the t direction. A terminal value problem with the boundary condition specified at $t = T$ can be propagated backwards in time from T, or by redefining the problem in terms of $\tau = T - t$ and propagating forward in τ.

The basic approach involves first rewriting the PDE in terms of V_t

$$V_t(S,t) = \rho(S)V(S,t) - \mu(S)V_S(S,t) - \tfrac{1}{2}\sigma^2(S)V_{SS}(S,t) - f(S) \qquad (1)$$

Given an initial condition $V(S,0) = B(S)$ we compute $V_S(S,0) = B'(S)$ and $V_{SS}(S,0) = B"(S)$ and thereby compute $V_t(S,0)$. This allows us to take a step of size Δt in the t direction and compute an approximation to $V(S,\Delta t)$. The algorithm then proceeds recursively. With a terminal condition we set $V(S,T) = B(S)$ and use a time step of $-\Delta t$; otherwise the problem is the same.

In finite difference implementations of this idea, a 2-dimensional grid is defined in the S and t directions and all derivatives terms are replaced by finite difference approximations (see Chapter ??). Using centered finite differences for the first and second derivatives for $O(h^2)$ accuracy, (1) becomes:

$$\begin{aligned} V_t(S_i,t) &= \rho(S_i)V(S_i,t) - \mu(S_i)\frac{V(S_{i+1},t) - V(S_{i-1},t)}{2h} \\ &\quad - \tfrac{1}{2}\sigma^2(S_i)\frac{V(S_{i+1},t) - 2V(S_i,t) + V(S_{i-1},t)}{h^2} \\ &\quad - f(S_i) + O(h^2) \\ &= \left[\frac{\mu(S_i)}{2h} - \frac{\sigma^2(S_i)}{2h^2}\right] V(S_{i-1},t) \\ &\quad + \left[\rho(S_i) + \frac{\sigma^2(S_i)}{h^2}\right] V(S_i,t) \\ &\quad + \left[-\frac{\mu(S_i)}{2h} - \frac{\sigma^2(S_i)}{2h^2}\right] V(S_{i+1},t) \\ &\quad - f(S_i) + O(h^2). \end{aligned}$$

Notice that the LHS of this expression depends on the value of the function at only three points, S_{i-1}, S_i and S_{i+1}.

We now must discretize the PDE in the t direction and it is here where we face several choices. The two obvious ones are to use either a forward difference

$$V_t(S_i,t) = \frac{1}{\Delta t}\Big(V(S_i,t+\Delta t) - V(S_i,t)\Big)$$

or a backward difference

$$V_t(S_i,t) = \frac{1}{\Delta t}\Big(V(S_i,t) - V(S_i,t-\Delta t)\Big)$$

Although it may initially appear that either will work, it turns out that the choice makes an enormous difference.

If we replace $V_t(S_i,t)$ in (2) with the forward difference operator and rearrange we get the system

$$\begin{aligned} V(S_i,t+\Delta t) &= a_i^f V(S_{i-1},t) + b_i^f V(S_i,t) + c_i^f V(S_{i+1},t) \\ &\quad - f(S_i)\Delta t + O(h^2) + O(\Delta t), \end{aligned}$$

where

$$\begin{aligned} a_i^f &= \mu(S_i)\frac{\Delta t}{2h} - \sigma^2(S_i)\frac{\Delta t}{2h^2} \\ b_i^f &= 1 + \rho(S_i)\Delta t + \sigma^2(S_i)\frac{\Delta t}{h^2} \\ c_i^f &= -\mu(S_i)\frac{\Delta t}{2h} - \sigma^2(S_i)\frac{\Delta t}{2h^2} \end{aligned}$$

On the other hand, if we use the backward difference operator we get

$$\begin{aligned} V(S_i,t-\Delta t) &= a_i^b V(S_{i-1},t) + b_i^b V(S_i,t) + c_i^b V(S_{i+1},t) \\ &\quad + f(S_i)\Delta t + O(h^2) + O(\Delta t), \end{aligned}$$

where

$$\begin{aligned} a_i^b &= -\mu(S_i)\frac{\Delta t}{2h} + \sigma^2(S_i)\frac{\Delta t}{2h^2} = -a_i^f \\ b_i^b &= 1 - \rho(S_i)\Delta t - \sigma^2(S_i)\frac{\Delta t}{h^2} = 2 - b_i^f \\ c_i^b &= \mu(S_i)\frac{\Delta t}{2h} + \sigma^2(S_i)\frac{\Delta t}{2h^2} = -c_i^f \end{aligned}$$

To implement the method we define a grid of m values of S_i ($S_1 = a, S_2 = a+h, \ldots, S_m = b$) and n values of t_j ($t_1 = 0, t_2 = \Delta t, \ldots, t_n = T$). For problems in which S is unbounded we must chose a and b such that the probability of attaining these values is very small. The grid can be represented as an $m \times n$ matrix with rows representing values of the state and columns representing points in time. Letting $V_{ij} = V(S_i,t_j)$ and given values of V_{1j} and V_{mj}, the forward difference system can be written in matrix form as the $m-2$ equation system

$$V_{j+1} = \begin{bmatrix} 1 & 0 & 0 & & & 0 & \\ a_2^f & b_2^f & c_2^f & & & & \\ 0 & a_3^f & b_3^f & c_3^f & & & \\ & & & \cdots & \cdots & \cdots & \\ & & & & a_{m-1}^f & b_{m-1}^f & c_{m-1}^f \\ 0 & & & & 0 & 0 & 1 \end{bmatrix} V_j - \begin{bmatrix} V_{1j} - V_{1j+1} \\ f_2\Delta t \\ f_3\Delta t \\ \cdots \\ f_{m-1}\Delta t \\ V_{mj} - V_{mj+1} \end{bmatrix}$$

whereas the backward difference system can be written

$$\begin{bmatrix} 1 & 0 & 0 & & & & 0 \\ a_2^b & b_2^b & c_2^b & & & & \\ 0 & a_3^b & b_3^b & c_3^b & & & \\ & & \cdots & \cdots & \cdots & & \\ & & & & a_{m-1}^b & b_{m-1}^b & c_{m-1}^b \\ 0 & & & & 0 & 0 & 1 \end{bmatrix} V_{j+1} = V_j + \begin{bmatrix} V_{1j+1} - V_{1j} \\ f_2 \Delta t \\ f_3 \Delta t \\ \cdots \\ f_{m-1} \Delta t \\ V_{mj+1} - V_{mj} \end{bmatrix}.$$

The first of these defines an explicit system of difference equations in V_{j+1} in terms of V_j. The second formulation defines V_{j+1} implicitly; obtaining an explicit solution requires solving the system of linear equations.

Notice that the endpoints S_1 and S_m are dealt with in a special way. The finite difference method requires that the solution be known at $S = a$ and $S = b$. We have written the first and last rows of the two linear systems to reflect that these values are known.

Although it may appear that the explicit approach is better, appearances can be deceiving. Although the explicit approach does not require a linear solve at each time step, it can be unstable when the time steps are not small enough. The instability arises because approximation errors are magnified as they are propagated through time, thus producing a useless result. Implicit methods are stable, regardless of the size of the time step, because the approximation errors are damped as they are propagated. Hence larger time steps can be used, resulting in greater computational efficiency. The trade-off is further tipped towards implicit methods by the fact that the linear system is sparse (indeed it is tridiagonal) and so special methods can be used to perform the linear solve.

In practice, a hybrid approach is often used. The Crank-Nicholson approach evaluates V, V_P, and V_{PP} as a weighted average of the finite difference approximations at time j and time $j+1$. Often the weights used are $\frac{1}{2}$ and $\frac{1}{2}$.

Unconditionally Stable Explicit Finite Difference Methods

The main disadvantage in using implicit finite difference methods is that the matrix inversion is relatively slow, even though the inverted matrix is tridiagonal. A stable explicit method could use considerably more evaluation points for the same computational time and thereby, hopefully, increase the accuracy of the approximation. There are a number of alternative explicit

methods that are unconditionally stable; we describe one here called the hopscotch method.

Hopscotch Method:

The hopscotch method alternately uses the explicit and implicit formulae for adjacent points. The idea is illustrated below:

i					
6	B	B	B	B	
5	I	X	I	B	
4	X	I	X	B	
3	I	X	I	B	
2	X	I	X	B	
1	I	X	I	B	
0	B	B	B	B	
	0	1	2	3	j

B: boundary values
X: explicitly determined values
I: implicitly determined values

In the figure there are 5 interior space points and 3 interior time points. Starting at time 2, one first calculates all of the points for which $(i + j)$ is even using the explicit scheme. Then one can calculate the points for which $(i + j)$ is odd using an implicit scheme that involves the previous computed spatial points on either side. This is continued for each successive time point. Thus for $(i + j)$ even, the value of $V[i, j]$ is computed using $V[i - 1, j - 1]$, $V[i, j - 1]$ and $V[i + 1, j - 1]$. For $(i + j)$ odd, on the other hand, $V[i, j]$ is computed using V[i,j-1], V[i-1,j] and $V[i + 1, j]$, both of the latter having already been computed explicitly.

Code Box 12.4: Hopscotch Method

Example: Financial Options
Finite difference methods are now routinely used to value financial options. Given a risk neutral process describing the value of an underlying asset:

$$dS = (r - \delta)Sdt + \sigma(S)dz,$$

an option value can be computed using the differential equation

$$rV = V_t + (r - \delta)SV_S + \tfrac{1}{2}\sigma^2(S)V_{SS}.$$

For European call options (no early exercise) with strike price k, the terminal (time T) boundary condition is

$$V(S, T) = \max(0, S - k),$$

the lower boundary condition is $V(0, t) = 0$ and the upper boundary condition is

$$V(S) = \exp(-\delta(T - t))S - \exp(-r(T - t))k.$$

For put options the boundary conditions are

$$V(S, T) = \max(0, k - S),$$

$$V(0, t) = \exp(-r(T - t))k$$

and $\lim S \to \infty\ V(S, t) = 0$.

Code Box 5 displays code that sets up the problem and calls the Hopscotch method to produce a solution for the case in which $\sigma(S) = \sigma S$ (geometric Brownian motion).

Table 12.1: Finite Difference Approximations for Linear Parabolic PDEs

$$V_t + \mu(x,t)V_S + \tfrac{1}{2}\sigma^2(S,t)V_{SS} = \rho(S,t)V$$

Forward Difference

$$\frac{V_{i,j+1} - V_{ij}}{\Delta t} + \mu_{ij}\frac{V_{i+1,j} - V_{i-1,j}}{2h} + \tfrac{1}{2}\sigma_{ij}^2\frac{V_{i+1,j} - 2V_{ij} + V_{i-1,j}}{2h^2} = \rho_{ij}V_{ij}$$

Backward Difference

$$\frac{V_{i,j+1} - V_{ij}}{\Delta t} + \mu_{ij}\frac{V_{i+1,j+1} - V_{i-1,j+1}}{2h} + \tfrac{1}{2}\sigma_{ij}^2\frac{V_{i+1,j+1} - 2V_{ij+1} + V_{i-1,j+1}}{2h^2} = \rho_{ij}V_{ij+1}$$

Note: if ρ , μ and σ are time dependent then define $\rho_{ij} = \rho(x_i, t_j + \frac{1}{2}\Delta t)$ and define μ_{ij} and σ_{ij} analogously.

Code Box 12.5: Computing Premia for Financial Options

12.1.2 Method of Lines for PDEs

Equation (2), which discretized a PDE in the state variable, defines an ODE in time. The method of lines recognizes this and uses any differential equation algorithm, such as the Runge-Kutta method, to solve the ODE.

Suppose instead of using finite difference approximations to the derivatives, we approximate the solution to the PDE as a weighted sum of a suitably chosen set of basis functions with time varying coefficients:

$$V(S,t) = \phi(S)c(t).$$

The PDE can then be written as

$$\phi(S)c'(t) = \left[\rho(S)\phi(S) - \mu(S)\phi'(S) - \tfrac{1}{2}\sigma^2(S)\phi''(S)\right] c(t) + f(S).$$

If we select a set of n nodes, S_i, and define the $n \times n$ basis matrix Φ, this has the form

$$c'(t) = Bc(t) + f,$$

where

$$B = \Phi^{-1}\left[\rho(S)\Phi\mu(S)\Phi'(S) - \tfrac{1}{2}\sigma^2(S)\Phi''(S)\right]$$

and

$$f = \Phi^{-1}f(S).$$

We thus have an ODE in the coefficients of the approximating function. Furthermore, for linear PDEs the associated ODE is linear and hence can be solved analytically in terms of the eigenvalues and eigenvectors of B.

[INCOMPLETE]

12.1.3 Collocation Approaches to Solving PDEs

The extended method of lines selects a set of basis functions and seeks to find an approximation at each point in time by solving a system of differential equations that the coefficients of the approximating function should satisfy. An alternative is define a set of basis functions for all of the variables and to determine the coefficients of an approximating function that satisfies the PDE and boundary conditions at a selected set of points. Not surprisingly, we will discuss the use of polynomial approximations and polynomial spline approximations. For linear PDEs, we will see that this approach leads to a relationship in the form

$$Bc = f,$$

where B is a basis matrix, c a vector of coefficients and f a vector. The coefficients are thus determined by solving a system of linear equations. For non-linear equations we have a more general

$$f(c; S, B) = 0,$$

where B here represents a set of basis matrices corresponding to the relevant partial derivatives.

Although finite difference methods are not associated with a specific set of basis functions, the finite difference operators can be viewed as defining "basis" matrices for the function and its derivatives. Thus, finite difference methods also lead to a relationship of the form $Bc = f$ for linear problems and $f(c; S, B)$ for non-linear problems.

[INCOMPLETE]

12.1.4 Variable Transformations

It is often useful in numerical (and analytical) analysis to transform problem variables. For example, suppose that a differential equation is defined over the domain $S \in [0, \infty)$. One way to handle this is to truncate the domain at some large value of S. An alternative is to transform the domain to a bounded range, e.g., to $[0, 1]$. Transformations will also prove useful in handling free (moving) boundary problems, which can be transformed to a domain with a constant boundary.

Transformations of PDEs can be a bit tricky so it is worth spending a little time discussing them in general and to provide some simple transformations

that are useful for numerical work. The transform changes the PDE to be solved; in particular, the derivative with respect to S must be rewritten in terms of z. In general, we have

$$V_S = V_z \frac{dz}{dS}$$

and

$$V_{SS} = V_z \frac{d^2 z}{dS^2} + V_{zz} \left(\frac{dz}{dS} \right)^2 .$$

A number of useful transformations for working with the various boundary conditions encountered in economics are summarized in Table 12.2. The first of these, $z = S/(c+S)$, transforms the domain from $[0, \infty)$ to $[0, 1]$. The parameter c is a scaling factor; values of S below c will map into the $[0, \frac{1}{2}]$ interval and values of S above c will map into the $[\frac{1}{2}, 1]$ interval. For this transform we have

$$V_S = V_z \frac{c}{(c+S)^2}$$

and

$$V_{SS} = V_z \frac{c^2}{(c+S)^4} - V_{zz} \frac{2c}{(c+S)^3}.$$

The transform is useful with problems in which the solution is known to be bounded as $S \to \infty$, such as the value of a put option. Using the transformation results in greater accuracy and eliminates the need to define the upper price level.[1]

The second transformation in Table 12.2 is useful with problems in which there is a non-zero lower boundary, possibly a free boundary, and a domain that is unbounded above (e.g., for American put options). The third transformation is useful in cases in which one or both of the boundaries are free boundaries. The final transformation listed is often used in models involving geometric Brownian motion ($dx = \mu x dt + \sigma x dz$). Although it makes a

[1]It can also work for valuing call option but both the call value and, possibly, the variance function (σ^2) can become unbounded as $z \to 1$. This can be addressed by defining the grid of z values to range from $z_0 = 0$ to $z_{n+1} = 1 - \Delta z$, where $\Delta z = n/(n+1)^2$. This seems to work well in practice, although extremely deep-in-the money call option may not be accurately valued (the practical importance of such inaccuracies is small).

problem that is bounded below into a doubly unbounded problem, it can be quite useful in cases in which the behavior at $S = 0$ is hard to capture.

To illustrate how these transformations are used consider the problem of numerically solving

$$\tfrac{1}{2}\sigma^2 S^2 V''(S) + \mu S V'(S) - rV(S) = 0$$

on $[a, \infty)$, with boundary conditions

$$V(a) = g_a$$

$$\lim_{S \to \infty} V(S) = g_b.$$

Using the second transformation on Table 12.2, $z = (S - a)/S$, the problem becomes

$$\tfrac{1}{2}\sigma^2 (1 - z)^2 v''(z) + (\mu - \sigma^2)(1 - z)v'(z) - rv(z) = 0$$

on $[0, 1]$, with boundary conditions

$$v(0) = g_a$$

$$v(1) = g_b.$$

To apply these notions to free boundary problems, consider a problem with a single state variable, defined on $[S^*, \infty)$, with two side conditions holding at S^*, say $V(S^*) = v_0$ and $V'(S^*) = v_1$. Consider the transformation from the interval $[S^*, \infty)$ to $[1, \infty)$ can be accomplished by using the transform

$$z = S/S^*.$$

With this transformation we can solve for V conditional on S^* and S^* conditional on V. In a two dimensional problem the free boundary is generally expressed as a functional relationship between the two states, say as

$$S_1 = g(S_2),$$

with the differential equation defined on $[g(S_2), \infty) \times [a, b]$. We can define a new state variable to replace the first state

$$z = S_1/g(S_2)$$

which will imply the following transformations

$$V(S_1, S_2) = v(z, S_2)$$

$$V_1(S_1, S_2) = v_1(z, S_2)/g(S_2)$$

$$V_{11}(S_1, S_2) = v_{11}(z, S_2)/g(S_2)^2$$

$$V_2(S_1, S_2) = v_2(z, S_2) - v_1 S_1 g'(S_2)/g(S_2)^2.$$

We use this strategy below to solve stochastic control problems.

12.2 Solving Stochastic Control Problems

In the previous chapter we saw that for problems of the form

$$V(S) = \max_{x(S)} \int_t^\infty e^{-r\tau} f(S, x) d\tau \text{ s.t. } dS = \mu(S, x)dt + \sigma(S)dz,$$

Bellman's equation takes the form

$$rV(S) = \max_{x(S)} \ f(S, x) + \mu(S, x)V'(S) + \tfrac{1}{2}\sigma^2(S)V''(S),$$

subject to boundary conditions at $S = a$ and $S = b$.

Suppose we approximate the function value function using $V(S) \approx \Phi(S)c$. For a given policy function $x(S)$, the collocation equations are

$$\left[r\phi(S) - \mu(s, x(S))\Phi'(S) - \tfrac{1}{2}\sigma^2(S, x(S))\Phi''(s)\right] c = f(S, x(S)).$$

Any relevant boundary conditions can be appended to the matrix in [ ]. For example, the value function may be known at the boundaries, in which case we have

$$\Phi(a)c = g_a \text{ and } \Phi(b)c = g_b.$$

The boundary conditions often are linear in the value function and its derivatives and hence are linear in the approximation coefficients. Given the linearity of the Bellman equation, the collocation equation is therefore linear in c and hence is easily solved.

An iterative procedure analogous to policy function iteration uses the following steps:

Table 12.2: Variable Transformations For First and Second Order Differential Equations

Transform	Domain/Codomain	$F'(S)$ $SF'(S)$	$F''(S)$ $S^2F''(S)$
$z = \frac{S}{c+S}$ $S = \frac{cz}{1-z}$	$[0,\infty) \to [0,1]$	$\frac{(1-z)^2}{c} f'(z)$ $z(1-z)f'(z)$	$\frac{(1-z)^3}{c^2}\Big((1-z)f''(z) - 2f'(z)\Big)$ $z^2(1-z)\Big((1-z)f''(z) - 2f'(z)\Big)$
$z = \frac{S-a}{S}$ $S = \frac{a}{1-z}$	$[a,\infty) \to [0,1]$	$\frac{(1-z)^2}{a} f'(z)$ $(1-z)f'(z)$	$\frac{(1-z)^3}{a^2}\Big((1-z)f''(z) - 2f'(z)\Big)$ $(1-z)\Big((1-z)f''(z) - 2f'(z)\Big)$
$z = \frac{S-a}{b-a}$ $S = a + (b-a)z$	$[a,b] \to [0,1]$	$\frac{f'(z)}{b-a}$ $\left(\frac{a}{b-a} + z\right) f'(z)$	$\frac{f''(z)}{(b-a)^2}$ $\left(\frac{a}{b-a} + z\right)^2 f''(z)$
$z = \frac{S}{a}$ $S = az$	$[a,\infty] \to [1,\infty]$	$\frac{f'(z)}{a}$ $zf'(z)$	$\frac{f''(z)}{a^2}$ $z^2 f''(z)$
$z = \ln(S)$ $S = e^z$	$[0,\infty) \to (-\infty,\infty)$	$e^{-z}f'(z)$ $f'(z)$	$e^{-2z}\Big(f''(z) - f'(z)\Big)$ $f''(z) - f'(z)$

A) guess an initial $V(S)$ and find the corresponding approximation coefficient vector c.

B) for each of the collocation nodes, S_i, determine the optimal value of x given the current value of c

C) solve the collocation equation for a new coefficient vector

D) check for convergence; return to (2) if not converged

An alternative when one can solve explicitly for the optimal control (in terms of the value function) is to substitute the control out of the Bellman Equation. This results in (generally) a nonlinear differential equation in S, which can be solved directly using collocation. If the differential equation is nonlinear, however, the collocation equations are also nonlinear and hence must be solved using a root finding algorithm.

As we discussed above, it may be useful to employ a change-of-variables to make the problem more tractable or more amenable to numerical approximation. For example, the value function may get very steep at $S = 0$, making it difficult to approximate using polynomial or spline bases. The optimal renewable resource harvesting example (Section ??) displays this behavior, with the limits of $V(0) = V_{SS}(0) = -\infty$ and $V_S(0) = \infty$. In such a case it is useful to employ the log transform (see Ludwig, 1979). The value function in the transformed variable is therefore approximately linear as the transformed state variable approaches $-\infty$; this is easily imposed on the transformed problem by forcing the second derivative to 0 at the lower boundary of the approximation. It is also reasonable to assume that the value becomes approximately linear as the state goes to ∞; in fact V_S should go to zero (the value of an additional unit of the resource is zero when the resource is infinitely abundant) implying that the value function approaches a constant. This can be insured by imposing that the transformed problem has a zero first or second derivative at the upper limit of the approximation. In practice, imposing the zero on the second derivative is preferred, as it allows for more flexibility in fit (this point is illustrated below).

Example: Harvesting a Renewable Resource

[INCOMPLETE]

12.2.1 Free Boundary Problems

Many of the problems discussed in the previous chapter involved free boundaries which represent endogenously determined state values at which some action is taken. For example, consider a second order linear differential equation with the general form

$$\rho(S)V(S) = f(S) + \mu(S)V'(S) + \tfrac{1}{2}\sigma^2(S)V_{SS}, \tag{2}$$

where this equation holds on some interval $[a, b]$. The usual boundary value problem takes both a and b as known and requires boundary conditions such as $V(a) = g_a$ and $V(b) = g_b$ to be met, where g_a and g_b are known values. Numerically, one can approximate the solution using a function parameterized by an n-vector c: $V(S) \approx \Phi(S)c$. c is chosen so that $\Phi(S)c$ satisfies (2) at $n-2$ points and satisfies the boundary conditions. This yields n equations in the n unknown parameters.

In the free boundary problem one or both of the boundary locations a and b are unknown and must be determined by satisfying some additional conditions. Suppose, for example that the upper boundary, b, is unknown but $V'(b) = h_b$, where h_b is a known constant. Thus there are three boundary conditions and one additional parameter, b, implying that one must solve $n+1$ equation in $n+1$ unknowns. If both boundaries are free, with $V'(a) = h_a$, the problem becomes one with $n+2$ equations and $n+2$ parameters.

The interval on which the approximating function is to be defined, however, is unknown. Fortunately, this problem is easily addressed using a change in variable. Consider first the case in which b is unknown and, for simplicity, $a = 0$. Define

$$z = S/b,$$

so the differential equation is defined on $z \in [0, 1]$. Define the function $v(z)$ such that

$$v(z) = V(S);$$

using the chain rule it can be seen that

$$v'(z) = V'(S)b$$

and

$$v''(z) = V''(S)b^2.$$

Inserting these definitions into (2) demonstrates that the original problem is equivalent to

$$\rho(bz)v(z) = f(bz) + \frac{\mu(bz)}{b}v'(z) + \frac{\sigma^2(bz)}{2b}v''(z), \tag{3}$$

for $z \in [0,1]$, with

$$v(0) = g_a,$$

$$v(1) = g_b,$$

and

$$v'(1) = h_b b.$$

Example: Timber Harvesting

[INCOMPLETE]

Optimal Stopping

Simple optimal stopping have the same form as the timber harvesting problem in that a PDE is solved over some unknown interval, where the endpoints of the interval must be determined along with the value function. We first illustrate this with a simple investment example.

More difficult stopping problems arise when the boundary changes over time, as is the case when solving American option pricing problems; American options can be exercised early and the free boundary represents the price, time values at which it is optimal to so. We illustrate one method for solving such problems; another method for solving such problem will be discussed in the context of stochastic bang-bang problems in the next section.

Example: Investment Under Uncertainty

Consider a simple irreversible investment problem in which an investment of I will generate a return stream with present value of S, where S is described by the Ito process

$$dS = \alpha(m - S)Sdt + \sigma S dz.$$

This process can be shown to have a mean reverting rate of return, with long-run mean m (see Section ??). When the investment is made it has value $S - I$. Prior to making the investment, however, the value of the right to make such an investment is $V(S)$, which is the solution to the following differential equation

$$\tfrac{1}{2}\sigma^2 S^2 V''(S) + \alpha(m - S)SV'(S) - rV(S) = 0,$$

where r is the risk-free interest rate. The lower boundary, $S = 0$, is associated with an investment value of 0, because once the process S goes to 0, it stays equal to 0 forever; hence $V(0) = 0$. The upper boundary is defined as the value, S^*, at which investment actually occurs. At this value two conditions must be met. The value matching condition states that at S^* the value of investing and not investing are equal: $V(S^*) = S^* - I$. The smooth-pasting optimality condition requires that $V'(S^*) = 1$.

Applying the change of variables ($z = S/S^*$) yields the equivalent problem

$$\tfrac{1}{2}\sigma^2 z^2 v''(z) + \alpha(m - zS^*)zv'(z) - rv(z) = 0, \tag{4}$$

on the interval $[0, 1]$, with $v(0) = 0$, $v(1) = S^* - I$, and $v'(1) = S^*$. To solve the problem we approximate the function $v(z)$ using

$$v(z, c) = \sum_{j=1}^{n} \phi_j(z)c_j,$$

where the $\phi_j(z)$ are convenient basis functions. Chebyshev polynomials are a natural choice for this problem because $v(z)$ should be relatively smooth. The parameter vector c and the optimal investment trigger S^* are selected to satisfy (4) at $n - 2$ appropriately chosen nodes on the interior of $[0, 1]$ (e.g., the roots of the order $n - 2$ Chebyshev polynomial) and to satisfy the three boundary conditions.

To make this a bit more explicit, given a guess of S^*, define the $n - 2 \times n$ matrix B

$$B_{ij} = \tfrac{1}{2}\sigma^2 z_i^2 \phi_j''(z_i) + \alpha(m - z_i S^*)z_i \phi_j'(z_i) - r\phi_j(z_i)$$

for $i = 1, \ldots, n - 2$. Then concatenate the basis functions for the boundary conditions to the bottom of this matrix: $B_{n-1,j} = \phi_j(0)$ and $B_{n,j} = \phi_j(1)$.

This produces an $n \times n$ matrix. The coefficients, conditional on the guess of S^*, are given by

$$c(S^*) = B^{-1} \begin{bmatrix} 0_{n-1} \\ S^* - I \end{bmatrix}.$$

Given c we can define a residual function in one dimension to solve for S^* using the smooth-parting condition:

$$r(S^*) = S^* - \phi'(1)c(S^*).$$

This approach works well in some cases but this example has one additional problem that must be addressed. It will be observed that, for some parameter values, the approximate solution obtained becomes unstable, exhibiting wide oscillations at low values of z. The solution value for S^*, however, remains reasonable. The problem, therefore, seems due to the approximation having trouble satisfying the lower boundary. It can be shown that, for some parameter values, the derivative of v becomes unbounded as S approaches 0:

$$\lim_{S \searrow 0} V'(S) = \infty.$$

This type of behavior cannot be well approximated by polynomials, the derivatives of which (at every order) are bounded on a bounded domain.

Fortunately this problem can be easily addressed by simply eliminating the lower boundary constraint and evaluating (4) at $n-1$ rather than $n-2$ nodes. This causes some error at very small values of z (or S) but does not cause significant problems at higher values of z. The economic context of the problem places far more importance on the values of z near 1, which defines the location of S^* and hence determines the optimal investment rule.

This particular problem has a partially known solution. It can be shown that the solution can be written as

$$V(S) = AS^{\beta} H(\phi S; \beta, \kappa),$$

where $H(x; \beta, \kappa)$ is the confluent hypergeometric function defined by the series expansion

$$H(x; \beta, \kappa) = \sum_{i=0}^{\infty} \frac{\Gamma(\beta + i)\Gamma(\kappa) x^i}{\Gamma(\beta)\Gamma(\kappa + i) i!}.$$

and

$$\beta = \frac{1}{2} - \frac{\alpha m}{\sigma^2} + \sqrt{\left(\tfrac{1}{2} - \frac{\alpha m}{\sigma^2}\right)^2 + \frac{2r}{\sigma^2}}$$

$$\kappa = 1 + 2\sqrt{\left(\tfrac{1}{2} - \frac{\alpha m}{\sigma^2}\right)^2 + \frac{2r}{\sigma^2}}$$

$$\phi = \frac{2\alpha}{\sigma^2}.$$

Thus, the problem can be seen to arise when $\beta < 1$, which causes the term in the derivative involving $S^{\beta-1}$ to become unbounded as $S \to 0$.

The solution is only partially known because the constants A and S^* must be determined numerically using the free boundary conditions:[2]

$$AS^{*\beta}H(\phi S^*;\beta,\kappa) - (S^* - I) = 0$$

and

$$A\beta S^{*\beta-1}H(\phi S^*;\beta,\kappa) + A\phi S^{*\beta}H'(\phi S^*;\beta,\kappa) - 1 = 0.$$

Eliminating A yields the relationship

$$S^* - \beta(S^* - I)\left(1 + \frac{\phi H(\phi S^*;\beta+1,\kappa+1)}{\kappa H(\phi S^*;\beta,\kappa)}S^*\right) = 0,$$

a simple root finding problem in a single variable, which can be solved using the methods of chapter ??.

MATLAB code solving the problem in both ways is shown below. This code produces Figure **??**. The dashed line is the solution obtained using the hypergeometric function approach. The dashed line solves the problem with no lower end point condition imposed and the dotted line imposes the lower end point condition. The figure illustrates the difficulties in fitting the value function at the lower end but also illustrates that the computation of the location of the free boundary is not very sensitive to these problems.

Example: Pricing American Options

[INCOMPLETE]

[2]Notice from the series expansion that the derivative of H is given by

$$H'(x;\beta,\kappa) = \frac{\beta}{\kappa}\sum_{i=0}^{\infty}\frac{\Gamma(\beta+i+1)\Gamma(\kappa)x^i}{\Gamma(\beta)\Gamma(\kappa+i+1)i!} = \frac{\beta}{\kappa}H(x;\beta+1,\kappa+1).$$

Stochastic Bang-Bang Problems

Problems with binary states that can be exited and reentered, as is the case with stochastic bang-bang problems, can lead to new challenges. These challenges arise because, in effect, two value functions, one for each of the binary states, must be simultaneously approximated. Furthermore, regions of the state space over which these value functions apply must be determined.

Recall that the general framework giving rise to stochastic bang-bang problems occurs when the reward function is of the form

$$f(S,x) = f_0(S) + f_1(S)x,$$

the state variable is governed by

$$dS = [g_0(S) + g_1(S)x]dt + \sigma(S)dz$$

and the control is bounded:

$$x_l \leq x \leq x_u.$$

Consider the discounted infinite time horizon problem

$$V(S) = \max_x E\left[\int_t^\infty e^{-\rho t} f(S,x)dt\right].$$

The optimal control is to set $x = x_l$ whenever $f_1 + g_1 V_S < 0$ and to set $x = x_u$ whenever $f_1 + g_1 V_S > 0$. Denoting these regions S_l and S_u, the value function must satisfy

$$\rho V - (g_0 + g_1 x_l) V_S - \tfrac{1}{2}\sigma^2 V_{SS} - (f_0 + f_1 x_l) = 0 \text{ on } S_l$$

$$\rho V - (g_0 + g_1 x_u) V_S - \tfrac{1}{2}\sigma^2 V_{SS} - (f_0 + f_1 x_u) = 0 \text{ on } S_u$$

and value-matching and smooth pasting at points were $f_1 = g_1 V_S$ (plus any additional boundary conditions at $S = a$ and $S = b$).

For concreteness suppose that there is a single point S^* such that $f_1(S^*) = g_1(S^*)V_S(S^*)$ and that S_l consists of points less than S^* and S_u of points greater than S^* (generally the context of the problem will suffice to determine the general nature of these sets). The numerical problem is to find this S^* and the value function $V(S)$. The following strategy can be used. First, notice that the Bellman equation is linear given S^* and assume that the boundary conditions are also linear in V. Suppose we approximate two functions, one

on S_l, the other on S_u that approximately satisfy the Bellman equations and the boundary conditions and also that, for any guess of S^*, satisfy value matching and smooth pasting at this guess.

Let the approximations be defined by $\phi(S)c_i$, for $i = l, u$ and define the function $B(S)$ as

$$B(S) = \rho\phi(S) - [g_0(S) + g_1(S)x_i]\,\phi'(S) - \tfrac{1}{2}\sigma^2(S)\phi''(S)$$

The c_i can be determined by making

$$B(S)c_i - [f_0(S) + f_1(S)x_i] = 0$$

at a selected set of collocation nodes, together with the boundary conditions and

$$\begin{array}{ll} \phi(S^*)c_l - \phi(S^*)c_u = 0 & \text{(value matching)} \\ \phi'(S^*)c_l - \phi'(S^*)c_u = 0 & \text{(smooth pasting).} \end{array}$$

Determining the c_i for some guess of S^*, therefore, amounts to solving a system of linear equations. Once the c_i are determined, the residual

$$r(S^*) = f_1(S^*) + g_1(S^*)V_S(S^*)$$

can be computed. The optimal value of S^* is then chosen to make $r(S^*) = 0$.

Example: Optimal Fish Harvest
Recall the optimal fish harvesting problem from Section 11.1.4. The value function solves the coupled PDE

$$\rho V = \begin{cases} \alpha S(1 - S/k)V_S + \frac{1}{2}\sigma^2 S^2 V_{SS} & \text{for } S < S^* \\ \pi E S + (\alpha S(1 - S/k) - ES)\,V_S + \frac{1}{2}\sigma^2 S^2 V_{SS} & \text{for } S > S^* \end{cases}$$

with S^* determined by $\pi = V_S(S^*)$ and continuity of V and V_S at S^*. For present purposes, impose the scale normalization $\pi = k = E = 1$ (by choosing scales for money, fish quantity and effort level).

It is a good idea to transform this problem by setting

$$z = \ln(S) - \ln(S^*).$$

This transformation has two effects: first, it simplifies the differential equation by making the coefficients constant or linear in S, and, second, it places the boundary between the two solution functions at $z = 0$.

The transformation necessitates rewriting the value function in terms of z, say as $v(z)$. The transformation implies that

$$S = S^* e^z,$$

$$v_z(z) = SV_S(S)$$

and

$$v_{zz}(z) = S^2 V_{SS}(S).$$

The transformed Bellman equation with the scale normalizations is

$$\rho v = \begin{cases} \left(\alpha(1 - S^* e^z) - \frac{1}{2}\sigma^2\right) v_z + \frac{1}{2}\sigma^2 v_{zz} & \text{for } z < 0 \\ S^* e^z + \left(\alpha(1 - S^* e^z) - \frac{1}{2}\sigma^2 - 1)\right) v_z + \frac{1}{2}\sigma^2 v_{zz} & \text{for } z > 0 \end{cases}.$$

It will be useful to rewrite this to isolate the S^* terms

$$\begin{array}{ll} \rho v - \left(\alpha - \frac{1}{2} s\sigma^2\right) v_z - \frac{1}{2}\sigma^2 v_{zz} + S^* \alpha e^z v_z = 0 & \text{for } z < 0 \\ \rho v - \left(\alpha - \frac{1}{2}\sigma^2 - 1\right) v_z - \frac{1}{2}\sigma^2 v_z z + S^* \alpha e^z v_z = S^* e^z & \text{for } z > 0 \end{array}.$$

The two functions are coupled by imposing continuity of v and v_z at $z = 0$. Technically there are also boundary conditions as z goes to $-\infty$ and ∞ , but we will ignore these for the time being.

Now let's approximate the two functions using $\phi_0(z)c_0$ and $\phi_1(z)c_1$, where the ϕ_i are n_i-element basis vectors and the c_i are the coefficients associated with these bases (not surprisingly, we will use Chebyshev polynomial bases). For a specific guess of S^*, the Bellman equation can be written

$$\begin{array}{ll} \left[\rho\phi_l(z) - \left(\alpha - \frac{1}{2} s\sigma^2\right) \phi_l'(z) - \frac{1}{2} s\sigma^2 \phi_l''(z)\right] c_l + S^* \left[\alpha e^z \phi_l'(z)\right] c_l = 0 & \text{for } z < 0 \\ \left[\rho\phi_u(z) - \left(\alpha - \frac{1}{2}\sigma^2 - 1\right) \phi_u'(z) - \frac{1}{2} s\sigma^2 \phi_u''(z)\right] c_u + S^* \left[\alpha e^z \phi_u'(z)\right] c_u = S^* e^z & \text{for } z > 0 \end{array}.$$

Evaluating this expression at a set of nodes, $z_l \in [a, 0]$, and $z_u \in [0, b]$, where a and b are arbitrary upper and lower bounds, with $a < 0$ and $b > 0$.

The boundary conditions at $z = 0$ for a given S^* are

$$\phi_l(0)c_l - \phi_u(0)c_u = 0$$

and

$$\phi_l'(0)c_l - \phi_u'(0)c_u = 0.$$

If we choose z_l and z_u to have $n_l - 1$ and $n_u - 1$ elements, respectively, this yields the $n_l + n_u$ system of linear equations:

$$\left(\begin{bmatrix} B_l & 0 \\ 0 & B_u \\ \phi_l(0) & -\phi_u(0) \\ \phi_l'(0) & -\phi_u'(0) \end{bmatrix} + S^* \begin{bmatrix} D_l & 0 \\ 0 & D_u \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \right) \begin{pmatrix} c_l \\ c_u \end{pmatrix} = \begin{pmatrix} 0 \\ S^* e^{z_u} \\ 0 \\ 0 \end{pmatrix}.$$

which has the form

$$(B + S^* D)c = S^* f.$$

The unknowns here are S^* (a scalar) and c (an $n_0 + n_1$ vector). The matrices B, D and f do not depend on either S^* or c; hence they can be predefined. Furthermore, this system of equations is linear in c and hence can be easily solved for a given S^*, thereby obtaining an approximation to the value function, v. We can therefore view c as a function of S^*:

$$c(S^*) = (B + S^* D)^{-1} S^* f.$$

The optimal S^* is then determined by solving the (non-linear) equation

$$S^* - \phi_l'(0) c_l(S^*) = 0.$$

It should be noted that the linearity in the coefficient vector c is not a special property of this problem; it arises from the linearity of the Bellman equation for a specified control function. We can summarize the approach in following way:

- Define the matrices B and D, both $(n_l + n_u) \times (n_l + n_u)$, and the vector f, $(n_l + n_u) \times 1$.
- Define a function that
  - accepts S^* as an argument,
  - computes $c(S^*)$,
  - returns $S^* - \phi_l'(0) c_l$
- Pass this function to a root finding solver.

A MATLAB implementation is displayed in Code Box 6. A script which computes and plots results is given in in Code Box 7; this was used to produce Figures 11.10, 11.11 and 11.12.

Code Box 12.6: Collocation File for Fish Harvesting Problem

Code Box 12.7: Script File for Fish Harvesting Problem

S^* is indicated in these plots with an '*'. Notice that the value function is continuous up to its second derivative, but that V'' exhibits a kink at $S = S^*$. This indicates why it is a good idea to break the value function apart and approximate it on each region separately, and pasting the two approximations together at the cut-off stock level. It also allows us to use the high degree of accuracy that polynomial approximations provide. One could, of course, approximate the entire value function with, say, a cubic spline, so long as you ensured that z=0 was a node. This would avoid the need to define two functions and thus has something to recommend it. However, it would require more nodes to achieve the same level of accuracy.

Example: Sequential Learning

In the previous example, the free boundary consisted of a single point. A more challenging boundary is required in the of learning-by-doing discussed in the previous chapter. This problem has the same form as that of the American option pricing problem. Here we provide an alternative framework for solving such problems.

Recall that the problem involved solving

$$rV = P - c(Q) + V_Q + (r - \delta)PV_P + \tfrac{1}{2}\sigma^2 P^2 V_{PP}$$

on $[P^*(Q), \infty) \times [0, Q_m]$, where $P^*(Q)$ is a free boundary to be determined. The boundary conditions are

$$P^*(Q)V_P(P^*(Q), Q) = \beta V(P^*(Q), Q),$$

$$P^*(Q)V_{PP}(P^*(Q), Q) = (\beta - 1)V_P(P^*(Q), Q)$$

$$V_{PP}(\infty, Q) = 0,$$

where β is the positive solution to

$$\tfrac{1}{2}\sigma^2\beta(\beta - 1) + (r - \delta)\beta - r = 0.$$

Also a terminal condition at $Q = Q_m$ is known and, for states below the free boundary, the value function is known up to a constant:

$$V(P, Q) = A(Q)P^\beta.$$

The difficulty with free boundaries is the unknown shape of the space over which the differential equation must hold. To get around this problem, we discuss a transformation method that regularizes the boundary. The PDE can then be solved over a rectangular region for a given guess of the location of the free boundary. An iterative root finding method is then applied to determine the position of the boundary. This method can be used with finite differences but it is better to use smoother approximations such as cubic splines or polynomials.

To illustrate the transformation method we define

$$z = \ln(P) - \ln(P^*(Q))$$

and will denote $v(x, Q) = V(P, Q)$. We are interested in solving the PDE for values of P on $[P^*(Q), \infty)$, which translates into values on z on $[0, \infty)$ (in practice we will typically truncate P). Given this transformation it is straightforward to verify the following relationships between the original and the transformed problem:

$$v_z(z, Q) = PV_P(P, Q)$$

$$v_{zz} - v_z = P^2 V_{PP}(P, Q)$$

and

$$V_Q = v_Q - \frac{P^{*'}(Q)}{P^*(Q)} v_z.$$

Substituting these expressions into the Bellman equation and the boundary conditions yields:

$$rv = P^* e^z - C(Q) + v_Q + (r - \delta - \tfrac{1}{2}\sigma^2 - P^{*'}/P^*)v_z + \tfrac{1}{2}\sigma^2 v_{zz},$$

$$v_z(0, Q) - \beta v(0, Q) = 0$$

$$v_{zz}(0, Q) - \beta v_z(0, Q) = 0$$

and

$$\lim_{z \to \infty} (v_{zz}(z, Q) - v_z(z, Q)) \exp(-2z) = 0.$$

One approach that can be used is to begin with an initial approximation to $P^*(Q)$. Use this approximation to obtain an approximate solution to

$$rv = P^* e^z - C(Q) + v_Q + (r - \delta - \tfrac{1}{2}\sigma^2 - P^{*'}/P^*)v_z + \tfrac{1}{2}\sigma^2 v_{zz},$$

$$v_z(0,Q) - \beta v(0,Q) = 0$$

and

$$\lim_{P\to\infty} (v_{zz}(P,Q) - v_z(P,Q))/P^2 = 0.$$

This is a linear problem and hence can be solved easily with the extended method of lines (treating Q as the "time" variable) or collocation in both P and Q.[3] The remaining boundary condition is then used to define a residual function

$$R(Q) = v_{zz}(0,Q) - \beta v_z(0,Q)$$

that is used to solve for the optimal $P^*(Q)$.

[INCOMPLETE]

[3]The collocation method has two drawbacks. First, it is an equilibrium method that does not utilize the propagation nature of the problem and therefore is slower than need be. Furthermore, the accuracy of the method is limited by the fact that, as the number of nodes is increased, the matrix operator used to define $v(z,Q)$ (given $P^*(Q)$) becomes very ill-conditioned. The ill-conditioning problem arises because of increasing large terms in the basis matrices for the derivatives. The largest term in the derivative basis is approximately equal to

$$\prod_{i=1}^{d} 4(n_i - 1)/(b_i - a_i)$$

while that of the second derivative basis is approximately

$$\prod_{i=1}^{d} 4(n_i - 1)^3/(b_i - a_i)^2$$

for a d-dimensional problem. Thus as the n_i rise, the largest element in the matrix rises as well.

Bibliographic Notes

The hopscotch method for solving PDEs is discussed by Ames, pp. 124-6.

The investment under uncertainty with mean reversion in the risk neutral return process is due to Dixit and Pindyck (pp. 161-163). We have simplified the notation by taking as given the risk-neutral process for the value of the completed investment. Their treatment took the actual value process as given and assumed that the required discount rate on the completed investment, ρ, is constant. This is equivalent to assuming that the market price of risk, θ, is a constant: $\theta(V) = (\rho - r)/\sigma$. It is also equivalent to assuming that the return stream, δ, generated by the completed investment is quadratic in V:

$$\delta(V) = (\rho - \alpha m)V + \alpha V^2.$$

The practical import of these assumption is to decrease the long-run mean of the value process by the amount $\theta\sigma/\alpha = (\rho - r)/\alpha$ when converting from the actual to the risk-neutral process.

The time-to-build exercise is from Madj and Pindyck.

Exercises

1. Modify the code in the fish harvesting example to compute the value function with a single cubic spline approximation. Plot the value function and its 1st and 2nd derivatives as functions of S (not z) and the residual function for the differential equation as a function of z. Be sure to include 0 as a node.

2. Consider the problem under the assumption that the effort (E) is not bounded (the problem thus becomes a barrier control problem). Write a program to solve for the value function and the optimal stock level that triggers harvesting. Use the same parameter values as in the bounded effort model ($\alpha = 0.1$, $\rho = 0.05$, $\sigma = 0.2$). Also compute and plot the optimal trigger stock level as a function of effort (E), using the above values for other parameters.

3. Cost Uncertainty

   Dixit and Pindyck (pp. 345-351) discuss the problem of determining an investment strategy when a project takes time to complete and completion costs are uncertain. The cost uncertainty takes two forms.

The first, technical uncertainty, arises because of unforeseen technical problems that develop as the project progresses. Technical uncertainty is assumed to be diversifiable and hence the market price of risk is zero. The second type of uncertainty is factor cost uncertainty, which is assumed to have market price of risk θ .

Define K to be the expected remaining cost to complete a project that is worth V upon completion. The dynamics of K are given by

$$dK = -Idt + \nu\sqrt{IK}dz + \gamma K dw,$$

where I, the control, is the current investment rate and dz and dw are independent Weiner processes. The project cannot be completed immediately because I is constrained by $0 \leq I \leq k$.Given the assumptions about the market price of risk, we convert the K process to its risk neutral form and use the risk free interest rate, r, to discount the future. Thus we act "as if"

$$dK = -(I + \theta\gamma K)dt + \nu\sqrt{IK}dz + \gamma K dw$$

and solve

$$F(K) = \max_{I(t)} E\left[e^{-rT}V - \int_0^T e^{-rt}I(t)dt\right],$$

where T is the (uncertain) completion time given by $K(T) = 0$.

The Bellman equation for this problem is

$$rF = \max_I -I - (I + \theta\gamma K)F'(K) + \tfrac{1}{2}(\nu^2 IK + \gamma^2 K^2)F''(K),$$

with boundary conditions

$$F(0) = V$$

$$F(\infty) = 0.$$

The optimal control is of the bang-bang type:

$$I = \begin{cases} 0 & \text{if } K > K^* \\ k & \text{if } K < K^* \end{cases}$$

where K^* solves

$$\tfrac{1}{2}\nu^2 F''(K) - F'(K) - 1 = 0.$$

Notice that technical uncertainty increases with the level of investment. This is a case in which the variance of the process is influenced by the control. Although we have not dealt with this explicitly, it raises no new problems.
a) Solve F up to an unknown constant for $K > K^*$.
b) Use the result in (a) to obtain a boundary condition at $K = K^*$ by utilizing the continuity of F and F'.
c) Solve the deterministic problem ($\nu = \gamma = 0$) and show that $K^* = k\ln(1 + rV/k)/r$.
d) Write the Bellman equation for $K < K^*$ and transform it from the domain $[0, K^*]$ to $[0, 1]$ using

$$z = K/K^*.$$

Also transform the boundary conditions.
e) Write a computer program using Chebyshev collocation to solve for F and K^* using the following parameters:

$$\begin{aligned} V &= 10 \\ r &= 0.05 \\ \theta &= 0 \\ k &= 2 \\ \gamma &= 0.5 \\ \nu &= 0.25. \end{aligned}$$

g) What alterations are needed to handle the case when $\gamma = 0$ and why are they needed.

4. Investment with Time-to-Build Constraints

Consider a situation in which an investment project, which upon completion will have a random value V and can be built by making a maximum current investment of k. Suppose that the value of the completed project evolves according to

$$dV = (\rho - \delta)Vdt + \sigma V dz,$$

where ρ is the return needed to compensate investors for the systematic risk associated with the project and $\delta = \rho - r$, where r is the risk free rate of return. The amount of investment needed to complete the project is K, which is a controlled process:

$$dK = -Idt.$$

In this situation it is optimal to either be investing at the maximum rate or not at all. Let the value of the investment opportunity in these two cases by denoted $F(V, K)$ and $f(V, K)$, respectively. These functions are governed by the following laws of motion:

$$\tfrac{1}{2}\sigma^2 V^2 F_{VV} + (r - \delta)VF_V - rF - kF_K - k = 0$$

and

$$\tfrac{1}{2}\sigma^2 V^2 f_{VV} + (r - \delta)Vf_V - rf = 0,$$

subject to the boundary conditions

$$F(V, 0) = V$$

$$\lim_{V \to \infty} F_V(V, K) = e^{-\delta K/k}$$

$$f(0, K) = 0$$

$$f(V^*, K) = F(V^*, K)$$

$$f_V(V^*, K) = F_V(V^*, K).$$

V^* is the value of the completed project needed to make a positive investment. It can be shown that $f(V) = A(K)V^\beta$, where

$$\beta = \frac{1}{2} - \frac{r - \delta}{\sigma^2} + \sqrt{\left(\tfrac{1}{2} - \frac{r - \delta}{\sigma^2}\right)^2 + \frac{2r}{\sigma^2}}. \tag{5}$$

and $A(K)$ is a function that must be determined by the boundary conditions. This may be eliminated by combining the free boundary conditions to yield

$$\beta F(V^*, K) = V^* F_V(V^*, K).$$

Summarizing, the problem is to solve the following partial differential equation for given values of σ, r, δ and k:

$$\tfrac{1}{2}\sigma^2 V^2 F_{VV} + (r-\delta)VF_V - rF - kF_K - k = 0,$$

subject to

$$F(V, 0) = V$$

$$\lim_{V\to\infty} F_V(V, K) = e^{-\delta K/k}$$

$$\beta F(V^*, K) = V^* F_V(V^*, K),$$

where β is given by (5). This is a PDE in V and K, with an initial condition for $K = 0$, a limiting boundary condition for large V and a lower free boundary for V that is a function of K.

Write MATLAB code to solve the time-to-build problem for the following parameter values:

$\delta = 0$
$r = 0.02$
$\sigma = 0.2$
$k = 1$

Appendix A

Mathematical Background

A.1 Normed Linear Spaces

A *linear space* or *vector space* is a nonempty set X endowed with two operations, vector addition $+$ and scalar multiplication $\cdot$, that satisfy

- $x + y = y + x$ for all $x, y \in X$
- $(x + y) + z = x + (y + z)$ for all $x, y, z \in X$
- there is a $\theta \in X$ such that $x + \theta = x$ for all $x \in X$
- for each $x \in X$ there is a $y \in X$ such that $x + y = \theta$
- $(\alpha\beta) \cdot x = \alpha \cdot (\beta \cdot x)$ for all $\alpha, \beta \in \Re$ and $x \in X$
- $\alpha \cdot (x + y) = \alpha \cdot x + \alpha \cdot y$ for all $\alpha \in \Re$ and $x, y \in X$
- $(\alpha + \beta) \cdot x = \alpha \cdot x + \beta \cdot y$ for all $\alpha, \beta \in \Re$ and $x \in X$
- $1 \cdot x = x$ for all $x \in X$.

The elements of X are called vectors.

A normed linear space is a linear space endowed with a real-valued function $||\cdot||$ on X, called a norm, which measures the size of vectors. By definition, a norm must satisfy

- $||x|| \geq 0$ for all $x \in X$;
- $||x|| = 0$ if and only if $x = \theta$;

- $||\alpha \cdot x|| = |\alpha| \; ||x||$ for all $\alpha \in \Re$ and $x \in X$;
- $||x + y|| \leq ||x|| + ||y||$ for all $x, y \in X$.

Every norm on a linear space induces a metric that measures the distance $d(x, y)$ between arbitrary vectors x and y. The induced metric is defined via the relation $d(x, y) = ||x - y||$. It meets all the conditions we normally expect a distance function to satisfy:

- $d(x, y) = d(y, x) \geq 0$ for all $x, y \in X$;
- $d(x, y) = 0$ if and only if $x = y \in X$;
- $d(x, y) \leq d(x, z) + d(z, y)$ for all $x, y, z \in X$.

Norms and metrics play a critical role in numerical analysis. In many numerical applications, we do not solve a model exactly, but rather compute an approximation via some iterative scheme. The iterative scheme is usually terminated when the change in successive iterates becomes acceptably small, as measured by the norm of the change. The accuracy of the approximation or approximation error is measured by the metric distance between the final approximant and the true solution. Of course, in all meaningful applications, the distance between the approximant and true solution is unknown because the true solution is unknown. However, in many theoretical and practical applications, it is possible to compute upper bounds on the approximation error, thus giving a level of confidence in the approximation.

In this book we will work almost exclusively with three classes of normed linear spaces. The first normed linear space is the familiar $\Re^n$, the space of all real n-vectors. The second normed linear space is $\Re^{m \times n}$, the space of all real m-by-n matrices. We will use a variety of norms for real vector and matrix spaces, all of which are discussed in greater detail in the following section.

The third class of normed linear space is $C(S)$, the space of all bounded continuous real-valued functions defined on $S \subset \Re^m$. Addition and scalar multiplication in this space are defined pointwise. Specifically, if $f, g \in C(S)$ and $\alpha \in \Re$, then $f + g$ is the function whose value at $x \in S$ is $f(x) + g(x)$ and αf is the function whose value at $x \in S$ is $\alpha f(x)$. We will use only one norm, called the sup or supremum norm, on the function space $C(S)$:

$$||f|| = \sup\{|f(x)| \mid x \in S\}.$$

In most applications, S will be a bounded interval of $\Re^n$.

A subset Y of a normed linear space X is called a subspace if it is closed under addition and scalar multiplication, and thus is a normed linear space in its own right. More specifically, Y is a subspace of X if $x + y \in Y$ and $\alpha x \in Y$ whenever $x, y \in Y$ and $\alpha \in \Re$. A subspace Y is said to be dense in X if for any $x \in X$ and $\epsilon > 0$, we can always find a $y \in Y$ such that $||x - y|| < \epsilon$. Dense linear subspaces play an important role in numerical analysis. When constructing approximants for elements in a normed linear space X, drawing our approximants from a dense linear subspace guarantees that an arbitrarily accurate approximation can always be found, at least in theory.

Given a nonempty subset S of X, span(S) is the set of all finite linear combinations of elements of S:

$$\text{span}(S) = \{\sum_{i=1}^{n} \alpha_i x_i | \alpha_i \in \Re, x_i \in X, n \text{ an integer}\}.$$

We say that a subset B is a basis for a subspace Y if $Y =$span(B) and if no proper subset of B has this property. A basis has the property that no element of the basis can be written as a linear combination of the other elements in the basis. That is, the elements of the basis are linearly independent.

Except for the trivial subspace $\{\theta\}$, a subspace Y will generally have many distinct bases. However, if Y has a basis with a finite number of elements, then all bases have the same number of nonzero elements and this number is called the dimension of the subspace. If the subspace has no finite basis, it is said to be infinite dimensional.

Consider some examples. Every normed linear space X, has two trivial subspaces: $\{\theta\}$, whose dimension is zero, and X. The sets $\{(0,1),(1,0)\}$ and $\{(2,1),(3,4)\}$ both are bases for $\Re^2$, which is a two-dimensional space; the set $\{(\alpha, 0.5 \cdot \alpha) | \alpha \in \Re\}$ is a one-dimensional subspace of $\Re^2$. In general, $\Re^n$ is an n-dimensional space with many possible bases; moreover, the span of any $k < n$ linearly independent n-vectors constitutes a proper k-dimensional subspace of $\Re^n$.

The function space $C(S)$ of all real-valued bounded continuous functions on an interval $S \subset \Re$ is an infinite-dimensional space. That is, there is no finite number of real-valued bounded continuous functions whose linear combinations span the entire space. This space has a number of subspaces that are important in numerical analysis. The set of all polynomials on S of

degree at most n forms an $n+1$ dimensional subspace of $C(S)$ with one basis being $\{1, x, x^2, \ldots, x^n\}$. The set of all polynomials, regardless of degree, is also a subspace of $C(S)$. It is infinite-dimensional. Other subspaces of $C(S)$ interest include the space of piecewise polynomials splines of a given order. These subspaces are finite-dimensional and are discussed further in the text.

A sequence $\{x_k\}$ in a normed linear space X converges to a limit x^* in X if $\lim_{k\longrightarrow\infty} ||x_k - x^*|| = 0$. We write $\lim_{k\longrightarrow\infty} x_k = x^*$ to indicate that the sequence $\{x_k\}$ converges to x^*. If a sequence converges, its limit is necessarily unique.

An open ball centered at $x \in X$ is a set of the form $\{y \in X \mid ||x-y|| < \epsilon\}$, where $\epsilon > 0$. A set S in X is open if every element of S is the center of some open ball contained entirely in S. A set S in X is closed if its complement, that is, the set of elements of X not contained in S, is an open set. Equivalently, a set S is closed if it contains the limit of every convergent sequence in S.

The Contraction Mapping Theorem has many uses in computational economics, particularly in existence and convergence theorems: Suppose that X is a complete normed linear space, that T maps a nonempty set $S \subset X$ into itself, and that, for some $\delta < 1$,

$$||T(x) - T(y)|| \leq \delta ||x - y||, \text{ for all } x, y \in S.$$

Then, there is an unique $x^* \in S$ such that $T(x^*) = x^*$. Moreover, if $x_0 \in S$ and $x_{k+1} = T(x_k)$, then $\{x_k\}$ necessarily converges to x^* and

$$||x_k - x^*|| \leq \frac{\delta}{1-\delta} ||x_k - x_{k-1}||.$$

When the above conditions hold, T is said to be a strong contraction on S and x^* is said to be a fixed-point of T in S.

We shall not define what we mean by a complete normed linear space, save to note that $\Re^n$, $C(S)$, and all their subspaces are complete.

A.2 Matrix Algebra

We write $x \in \Re^n$ to denote that x is an n-vector whose i^{th} entry is x_i. A vector is understood to be in column form unless otherwise noted.

If x and y are n-vectors, then their sum $z = x + y$ is the n-vector whose i^{th} entry is $z_i = x_i + y_i$. Their inner product or dot product, $x \star y$, is the real

number $\sum_i x_i \cdot y_i$. And their array product, $z = x. \star y$, is the n-vector whose i^{th} entry is $z_i = x_i \cdot y_i$.

If α is a scalar, that is, a real number, and x is an n-vector, then their scalar sum $z = \alpha + x = x + \alpha$ is the n vector whose i^{th} entry is $z_i = \alpha + x_i$. Their scalar product, $z = \alpha \star x = x \star \alpha$, is the n-vector whose i^{th} entry is $z_i = \alpha \cdot x_i$.

The most useful vector norms are, respectively, the 1-norm or sum norm, the 2-norm or Euclidean norm, and the infinity or sup norm:

$$\begin{aligned} ||x||_1 &= \textstyle\sum_i |x_i|, \\ ||x||_2 &= \sqrt{\textstyle\sum_i |x_i|^2}, \\ ||x||_\infty &= \max\{|x_1|, |x_2|, \ldots, |x_n|\}. \end{aligned}$$

In Matlab, the norms may be computed for any vector x, respectively, by writing: `norm(x,1)`, `norm(x,2)`, and `norm(x,inf)`. If we simply write `norm(x)`, the 2-norm or Euclidean norm is computed.

All norms on $\Re^n$ are equivalent in the sense that a sequence converges in one vector norm, if and only if it converges in all other vector norms. This is not true of generally of all normed linear spaces.

A sequence of vectors $\{x_k\}$ converges to x^* at a rate of order $p \geq 1$ if for some $c \geq 0$ and for sufficiently large n,

$$||x_{k+1} - x^*|| \leq c||x_k - x^*||^p.$$

If $p = 1$ and $c < 1$ we say the convergence is linear; if $p > 1$ we say the convergence is superlinear; and if $p = 2$ we say the convergence is quadratic.

We write $A \in \Re^{m \times n}$ to denote that A is an m-row by n-column matrix whose row i, column j entry, or, more succinctly, ij^{th} entry, is A_{ij}.

If A is an m by n matrix and B is an m by n matrix, then their sum $C = A + B$ is the m by n matrix whose ij^{th} entry is $C_{ij} = A_{ij} + B_{ij}$. If A is an m by p matrix and B is a p by n matrix, then their product $C = A \star B$ is the m by n matrix whose ij^{th} entry is $C_{ij} = \sum_{k=1}^{p} A_{ik} B_{kj}$. If A and B are both m by n matrices, then their array product $C = A. \star B$ is the m by n matrix whose ij^{th} entry is $C_{ij} = A_{ij} B_{ij}$.

A matrix A is square if it has an equal number of rows and columns. A square matrix is upper triangular if $A_{ij} = 0$ for $i > j$; it is lower triangular if $A_{ij} = 0$ for $i < j$; it is diagonal if $A_{ij} = 0$ for $i \neq j$; and it is tridiagonal if $A_{ij} = 0$ for $|i - j| > 1$. The identity matrix, denoted I, is a diagonal matrix whose diagonal entries are all 1. In Matlab, the identity matrix of order n may is generated by the statement `eye(n)`.

The transpose of an m by n matrix A, denoted A', is the n by m matrix whose ij^{th} entry is the ji^{th} entry of A. A square matrix is symmetric if $A = A'$, that is, if $A_{ij} = A_{ji}$ for all i and j. A square matrix A is orthogonal if $A' \star A = A \star A'$ is diagonal, and orthonormal if $A' \star A = A \star A' = I$. In Matlab, the transpose of a matrix A is generated by the statement A'.

A square matrix A is invertible if there exists a matrix A^{-1}, called the inverse of A, such that $A \star A^{-1} = A^{-1} \star A = I$. If the inverse exists, it is unique. In Matlab, the inverse of a square matrix A can be generated by the statement `inv(A)`.

The most useful matrix norms, and the only ones used in this book, are constructed from vector norms. A given n-vector norm $||\cdot||$ induces a corresponding matrix norm for n by n matrices via the relation

$$||A|| = \max_{||x||=1} ||A \star x||$$

or, equivalently,

$$||A|| = \max_{||x|| \neq 0} \frac{||A \star x||}{||x||}.$$

Given corresponding vector and matrix norms,

$$||A \star x|| \leq ||A|| \cdot ||x||.$$

Moreover, if A and B are square matrices,

$$||A \star B|| \leq ||A|| \cdot ||B||.$$

Common matrix norms include the matrix norms induced by the sum, Euclidean, and sup norms:

$$||A||_p = \max_{||x||_p=1} ||A \star x||_p$$

for $p = 1, 2, \infty$. In Matlab, these norms may be computed for any matrix A, respectively, by writing: `norm(A,1)`, `norm(A,2)`, and `norm(A,inf)`. The Euclidean matrix norm is relatively expensive to compute. The sum and sup norms, on the other hand, take a relatively simple form:

$$\begin{aligned} ||A||_1 &= \max_{1 \leq j \leq n} \sum_{i=1}^{n} |A_{ij}| \\ ||A||_\infty &= \max_{1 \leq i \leq n} \sum_{j=1}^{n} |A_{ij}|. \end{aligned}$$

The spectral radius of a square matrix A, denoted $\rho(A)$, is the infimum of all the matrix norms of A. We have $\lim_{k=1}^{\infty} A^k = 0$ if and only if $\rho(A) < 1$, in which case $(I - A)^{-1} = \sum_{k=1}^{\infty} A^k$. Thus, if $||A|| < 1$ in any vector norm, A^k converges to zero.

A square symmetric matrix A is negative semidefinite if $x' \star A \star x \leq 0$ for all x; it is negative definite if $x' \star A \star x < 0$ for all $x \neq 0$; it is positive semidefinite if $x' \star A \star x \geq 0$ for all x; and it is positive definite if $x' \star A \star x > 0$ for all $x \neq 0$.

A.3 Real Analysis

The gradient or Jacobian of a vector-valued function $f : \Re^n \mapsto \Re^m$ is the m by n matrix-valued function of first partial derivatives of f. More specifically, the gradient of f at x, denoted by either $f'(x)$ or $f_x(x)$, is the m by n matrix whose ij^{th} entry is the partial derivative $\frac{\partial f_i}{\partial x_j}(x)$. More generally, if $f(x_1, x_2)$ is an n-vector-valued function defined for $x_1 \in \Re^{n_1}$ and $x_2 \in \Re^{n_2}$, then $f_{x_1}(x)$ is the m by n_1 matrix of partial derivatives of f with respect to x_1 and $f_{x_2}(x)$ is the m by n_2 matrix of partial derivatives of f with respect to x_2.

The Hessian of the real-valued function $f : \Re^n \mapsto \Re$ is the n by n matrix-valued function of second partial derivatives of f. More specifically, the Hessian of f at x, denoted by either $f''(x)$ or $f_{xx}(x)$, is the symmetric n by n matrix whose ij^{th} entry is $\frac{\partial^2 f}{\partial x_i \partial x_j}(x)$. More generally, if $f(x_1, x_2)$ is a real-valued function defined for $x_1 \in \Re^{n_1}$ and $x_2 \in \Re^{n_2}$, where $n_1 + n_2 = n$, then $f_{x_i x_j}(x)$ is the n_i by n_j submatrix of $f''(x)$ obtained by extracting the rows corresponding to the elements of x_i and the columns corresponding to the columns of x_j.

A real-valued function $f : \Re^n \mapsto \Re$ is smooth on a convex open set S if its gradient and Hessian are defined and continuous on S. By Taylor's theorem, a smooth function may be approximated locally by either a linear or quadratic function. More specifically, for all x in S,

$$f(x) = f(x_0) + f_x(x_0) \star (x - x_0) + o(||x - x_0||)$$

and

$$\begin{aligned} f(x) = {} & f(x_0) + f_x(x_0) \star (x - x_0) \\ & + \tfrac{1}{2}(x - x_0)' \star f_{xx}(x_0) \star (x - x_0) + o(||x - x_0||^2) \end{aligned}$$

where $o(t)$ denotes a term with the property that $lim_{t \longrightarrow 0}(o(t)/t) = 0$.

The Intermediate Value Theorem asserts that if a continuous real-valued function attains two values, then it must attain all values in between. More precisely, if f continuous on a convex set $S \in \Re^n$ and $f(x_1) \leq y \leq f(x_2)$ for some $x_1 \in S$, $x_2 \in S$, and $y \in \Re$, then $f(x) = y$ for some $x \in S$.

The Implicit Function Theorem gives conditions under which a system of nonlinear equations will have a locally unique solution that will vary continuously with some parameter: Suppose $F : \Re^{m+n} \mapsto \Re^n$ is continuously differentiable in a neighborhood of (x_0, y_0), $x_0 \in \Re^m$ and $y_0 \in \Re^n$, and that $F(x_0, y_0) = 0$. If $F_y(x_0, y_0)$ is nonsingular, then there is an unique function $f : \Re^m \mapsto \Re^n$ defined on a neighborhood N of x_0 such that for all $x \in N$, $F(x, f(x)) = 0$. Furthermore, the function f is continuously differentiable on N and $f'(x) = -F_y^{-1}(x, f(x)) \star F_x(x, f(x))$.

A subset S is bounded if it is contained entirely inside some ball centered at zero. A subset S is compact if it is both closed and bounded. A continuous real-valued function defined on a compact set has well-defined maximum and minimum values; moreover, there will be points in S at which the function attains its maximum and minimum values.

A real-valued function $f : \Re^n \mapsto \Re$ is concave on a convex set S if $\alpha_1 f(x_1) + \alpha_2 f(x_2) \leq f(\alpha_1 x_1 + \alpha_2 x_2)$ for all $x_1, x_2 \in S$ and $\alpha_1, \alpha_2 \geq 0$ with $\alpha_1 + \alpha_2 = 1$. It is strictly concave if the inequality is always strict. A smooth function is concave (strictly concave) if and only if $f''(x)$ is negative semidefinite (negative definite) for all $x \in S$. A smooth function f is convex if and only $-f$ is concave. If a function is concave (convex) on an convex set, then its maximum (minimum), if it exists, is unique.

A.4 Markov Chains

A Markov process is a sequence of random variables $\{X_t \mid t = 0, 1, 2, \ldots\}$ with common state space S whose distributions satisfy

$$\Pr\{X_{t+1} \in A \mid X_t, X_{t-1}, X_{t-2}, \ldots\} = \Pr\{X_{t+1} \in A \mid X_t\} \quad A \subset S.$$

A Markov process is often said to be memoryless because the distribution X_{t+1} conditional on the history of the process through time t is completely determined by X_t and is independent of the realizations of the process prior to time t.

A Markov chain is a Markov process with a finite state-space $S = \{1, 2, 3, \ldots, n\}$.

A Markov chain is completely characterized by its transition probabilities

$$P_{tij} = \Pr\{X_{t+1} = j \mid X_t = i\}, \qquad i, j \in S.$$

A Markov chain is stationary if its transition probabilities

$$P_{ij} = \Pr\{X_{t+1} = j \mid X_t = i\}, \qquad i, j \in S$$

are independent of t. The matrix P, called the transition probability matrix.

The steady-state distribution of a stationary Markov chain is a probability distribution $\{\pi_i | i = 1, 2, \ldots, n\}$ on S, such that

$$\pi_j = \lim_{\tau \to \infty} \Pr\{X_\tau = j \mid X_t = i\} \qquad i, j \in S.$$

The steady-state distribution π, if it exists, completely characterizes the longrun behavior of a stationary Markov chain.

A stationary Markov chain is irreducible if for any $i, j \in S$ there is some $k \geq 1$ such that $\Pr\{X_{t+k} = j \mid X_t = i\} > 0$, that is, if starting from any state there is positive probability of eventually visiting every other state. Given an irreducible Markov chain with transition probability matrix P, if there is an n-vector $\pi \geq 0$ such that

$$\begin{aligned} P' \star \pi &= \pi \\ \textstyle\sum_i \pi_i &= 1, \end{aligned}$$

then the Markov chain has a steady-state distribution π.

In computational economic applications, one often encounters irreducible Markov chains. To compute the steady-state distribution of the Markov chain, one solves the $n + 1$ by n linear equation system

$$\begin{bmatrix} I - P' \\ i' \end{bmatrix} \star \pi = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

where P is the probability transition matrix and i is the vector consisting of all ones. Due to linear dependency among the probabilities, any one of the first n linear equations is redundant and may be dropped to obtain an uniquely soluble matrix linear equation.

Consider a stationary Markov chain with transition probability matrix

$$P = \begin{bmatrix} 0.5 & 0.2 & 0.3 \\ 0.0 & 0.4 & 0.6 \\ 0.5 & 0.5 & 0.0 \end{bmatrix}$$

Although one cannot reach state 1 from state 2 in one step, one can reach it with positive probability in two steps. Similarly, although one cannot return to state 3 in one step, one can return in two steps. The steady-state distribution π of the Markov chain may be computed by solving the linear equation

$$\begin{bmatrix} 0.5 & 0.0 & -0.5 \\ -0.2 & 0.6 & -0.5 \\ 1.0 & 1.0 & 1.0 \end{bmatrix} \star \pi = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

The solution is

$$\pi = \begin{bmatrix} 0.316 \\ 0.368 \\ 0.316 \end{bmatrix}.$$

Thus, over the long run, the Markov process will spend about 32.6 percent of its time in state 1, 36.8 percent of its time in state 2, and 31.6 percent of its time in state 3.

Appendix B

Computer Programming

B.1 Computer Arithmetic

Some knowledge of how computers perform numerical computations and how programming languages work is useful in applied numerical work, especially if one is to write efficient programs. It often comes as an unpleasant surprise to many people to learn that exact arithmetic and computer arithmetic do not always give the same answers, even in programs without programming errors.

For example, consider the following two statements

```
x = (1e-20 + 1) - 1
```

and

```
x = 1e-20 + (1 - 1).
```

Here, 1e-20 is computer shorthand for 10^{-20}. Mathematically the two statements are equivalent because addition and subtraction are associative. A computer, however, would evaluate these statements differently. The first statement would likely result in $x = 0$, whereas the second would result in $x = 1e - 20$. The reason has to do with how computers represent numbers.

Typically, computer languages such as Fortran and C allow several ways of representing a number. Matlab makes things simple by only have one representation for a number. Matlab uses what is often called a double precision floating point number. The exact details of the representation depends on the hardware but there are several features in common. First, the representation has three parts, a sign bit, an exponent, a mantissa. Consider the

number -3210.4. This can be equivalently written as -3.2104×10^3. The mantissa is 3.2104, the exponent is 3, and the sign bit is 1.

The computer has only a predefined set of storage elements (bytes) for a number. On most personal computers a double precision number has 8 bytes. If the mantissa is very long it gets truncated by rounding or chopping, depending on the hardware. For example, suppose only 5 places are allocated for the mantissa. A number like -3210.48 might be represented as -3.2104×10^3, that is, the lowest digit may be chopped off.

In our original example, when the computer processes $x = (1e-20+1)-1$ it first adds 1 to 1e-20, which is the number 1.00000000000000000001. Unfortunately, most computers cannot handle this long a mantissa and truncate the result to 1. The computer then subtracts 1 from the first sum, which results in 0. On the other hand, with the statement $x = 1e-20+(1-1)$, the subtraction in parenthesis occurs first, resulting in 0, which is then added to $1e-20$.

To understand more fully how numbers are stored in a computer, let us examine a few numbers in their so-called hexadecimal form. Hexadecimal numbers are numbers expressed in base 16; this is a useful base for computer arithmetic because it is a power of base 2, which is the form in which numbers are ultimately stored in a computer.[1] Hexadecimal numbers use the usual digits 0 through 9 and supplement them with the letters a through f; a=10, b=11,..., f=15. An 8-byte floating point number (i.e., "double-precision") looks something like:

```
3ff1 0000 0000 0000.
```

The hexadecimal representation makes clear some of the problems that arise in floating point arithmetic. Suppose one compared the values derived by the following expressions

$$1/3 + 1/2$$

and

$$5/6.$$

The first operation results in

```
3fea aaaa aaaa aaaa
```

[1]You can see the hexadecimal representation of a number in MATLAB by using `format hex`.

whereas the second results in

`3fea aaaa aaaa aaab`.

We know that these operations should result in the same number but the computer represents them in a way that differs by a single bit in the lowest order byte. Although this may not seem like a big deal, if one were to test the expression

```
1/3+1/2==5/6
```

the expression would be deemed false.

Similar problems arise in other case as well. For example,

$7^{-20} =$ `3c6c e5e8 5616 4656`

whereas

$7^{-19}/7 =$ `3c6c e5e8 5616 4655`,

even though in exact arithmetic these two quantities are theoretically the same.

Reversing a mathematical operation sometimes does not work either. In general, one should be careful when a number is raised to a large power and then to a very small power or vice versa. For example,

$$(1.1^(10e-12))^(10e+12)$$

should result in 1.1. However, on many computers the operation will result in 1.09941652517756.

Roundoff error is only one of the pitfalls of computer programming. In numerical computations, error is also introduced by the computer's inherent inability to evaluate certain mathematical expressions exactly. For all its power, a computer can only perform a limited set of arithmetic operations directly. Essentially this list includes the four arithmetic operations of addition, subtraction, multiplication and division, as well as logical operations of comparison. Other common functions, such as exponential, logarithmic, and trigonometric functions cannot be evaluated directly using computer arithmetic. They can only be evaluated approximately using algorithms based on the four basic arithmetic operations.

For the common functions very efficient algorithms typically exist and these are sometimes "hardwired" into the computer's processor or coprocessor. An important area of numerical analysis involves determining efficient

approximations that can be computed using basic arithmetic operations. For example, the exponential function has the series representation

$$\exp(x) = \sum_{i=0}^{\infty} x^n/n!.$$

Obviously one cannot compute the infinite sum, but one could compute a finite number of these terms, with the hope that one will obtain sufficient accuracy for the purpose at hand. The result, however, will always be inexact.

B.2 Data Storage

Matlab's basic data type is the matrix, with a scalar just a 1 by 1 matrix and an n-vector an n by 1 or 1 by n matrix. Actually, the basic data type in Matlab also contains additional information that is stored along with the matrix itself. In particular, Matlab attaches the row and column information about the matrix. This is a significant advantage over writing in low level language like Fortran or C because it relieves one of the necessity of keeping track of array size and memory allocation.

When one wants to represent an m by n matrix of numbers in a computer there are a number of ways to do this. The most simple way is to store all the elements sequentially in memory, starting with the one indexed (1,1) and working down successive columns or across successive rows until the (m,n)th element is stored. Different languages make different choices about how to store a matrix. Fortran stores matrices in column order, whereas C stores in row order. Matlab, although written in C, stores in column order, thereby conforming with the Fortran standard.

Many matrices encountered in practice are sparse, meaning that they consist mostly of zero entries. Clearly, it is a waste of memory to store all of the zeros, and it is time consuming to process the zeros in arithmetic matrix operations. Matlab allows one to store a sparse matrix efficiently by keeping track of only the non-zero elements of the original matrix and their location. In this storage scheme, the row indices and non-zero entries are stored in a two-column vector. A separate vector is used to keep track of where the first element in each column is located. If one wants to access element (i, j), Matlab checks the jth element of the column indicator vector to find where the jth column starts and then searches the row column for the ith element (if one is not found then the element must be zero).

Although sparse matrix representations are useful, their use incurs a cost. To access element (i, j) of a full matrix, one simply goes to element (i-1)*m+j storage location the. To access an element in a sparse matrix involves a search over row indices and hence can take longer. This additional overhead can add up significantly and actually slow down a computational procedure.

A further consideration in using sparse matrices concerns memory allocation. If a procedure repeatedly alters the contents of a sparse matrix, the memory needed to store the matrix may change, even if its dimension does not. This means that more memory may be needed each time the number of non-zero elements increases. This memory allocation is both time consuming and may eventually exhaust computer memory. This problem does not arise with full matrices because mn elements are stored in fixed locations from the beginning.

The decision whether to use a sparse or full matrix representation depends on a balance between a number of factors. Clearly for very sparse matrices (less than 10% non-zero) one is better off using sparse matrices and anything over 67% non-zeros one is better off with full matrices (which actually require less storage space at that point). In between, some experimentation may be required to determine which is better for a given application.

B.3 Programming Style

In general there are different ways to write a program that produce the same end results. Algorithmic efficiency refers to the execution time and memory used to get the job done. In many cases, especially in a matrix processing language like Matlab, there are important trade-offs between execution time and memory use. Often, however, the trade-offs are trivial and one way of writing the code may be unambiguously better than another.

In Matlab, the rule of thumb is to avoid loops where possible. Matlab is a hybrid language that is both interpreted and complied. A loop executed by the interpreter is generally slower than direct vector operations that are implemented in compiler code. For example, suppose one had a scalar x that one wanted to multiply by the integers from 1 to n to create a vector y whose i^{th} entry is $y_i = x^i$. Both of the following code segments produce the desired result:

```
for k=1:n
  y(i)=x^i;
end
```

and

```
y=x.^(1:n);
```

The second way avoids the looping of the first and hence executes substantially faster.

Programmer development effort is another critical resource required in program construction that is sometimes ignored in discussions of efficiency. One reason for using high level language such as Matlab, rather than a low level language such as Fortran, is that programming time is often greatly reduced. Matlab carries out many of the housekeeping tasks that the programmer must deal with in lower level languages. Even in Matlab, however, one should consider carefully how important it is to write very efficient code. If the code will be used infrequently, less effort should be devoted to making the code computationally efficient than if the code will be used often or repeatedly.

Furthermore, computationally efficient code can sometimes be fairly difficult to read. If one plans to revise the code at a later date or if someone else is going to use it, it may be better to approach the problem in a simpler way that is more transparent, though possibly slower. The proper balance of computational efficiency versus clarity and development effort is a judgment call. A good idea, however, is embodied in the saying "Get it to run right, then get it to run fast." In other words, get one's code to do what one what it to do first, then look for ways to improve its efficiency.

It is especially important to document one's code. It does not take long for even an experienced programmer to forget what a piece of code does if it is undocumented. We suggest that one get in the habit of writing headers that explain clearly what the code in a file does. If it is a function, the header should contain details on the input and output arguments and on the algorithm used (as appropriate), including references. Within the code it is a good idea to sprinkle reminders about what the code is doing at that point.

Another good programming practice is modularity. Functions that perform a simple well defined task that is to be repeated often should be written separately and called from other functions as needed. The simple functions can be debugged and then depended on to perform their job in a variety of applications. This not only saves program development time, but makes the resulting code far easier to understand. Also, if one decides that there is a better way to write such a function, one need only make the changes in one place. An example of this principle is a function that computes the deriva-

tives of a function numerically. Such a function will be used extensively in this book.