# Applied Computational Economics and Finance

Mario J. Miranda
and Paul L. Fackler

# Applied Computational Economics and Finance

Mario J. Miranda
The Ohio State University

and

Paul L. Fackler
North Carolina State University

# Contents

**Preface** **xii**

**1 Introduction** **1**
1.1 Some Apparently Simple Questions . . . . . 1
1.2 An Alternative Analytic Framework . . . . . 3
Exercises . . . . . 5

**2 Computer Basics and Linear Equations** **8**
2.1 Computer Arithmetic . . . . . 8
2.2 Data Storage . . . . . 12
2.3 Linear Equations and the L-U Factorization . . . . . 13
2.4 Gaussian Elimination . . . . . 17
2.5 Rounding Error . . . . . 19
2.6 Ill Conditioning . . . . . 20
2.7 Special Linear Equations . . . . . 22
2.8 Iterative Methods . . . . . 23
Exercises . . . . . 25
Bibliographic Notes . . . . . 28

**3 Nonlinear Equations** **30**
3.1 Bisection Method . . . . . 31
3.2 Function Iteration . . . . . 33
3.3 Newton's Method . . . . . 35
3.4 Quasi-Newton Methods . . . . . 39
3.5 Problems With Newton Methods . . . . . 43
3.6 Choosing a Solution Method . . . . . 46
3.7 Complementarity Problems . . . . . 48
3.8 Complementarity Methods . . . . . 51
Exercises . . . . . 55
Bibliographic Notes . . . . . 62

**4 Finite-Dimensional Optimization** **63**
4.1 Derivative-Free Methods . . . 64
4.2 Newton-Raphson Method . . . 68
4.3 Quasi-Newton Methods . . . 70
4.4 Line Search Methods . . . 74
4.5 Special Cases . . . 77
4.6 Constrained Optimization . . . 79
Exercises . . . 83
Bibliographic Notes . . . 93

**5 Integration and Differentiation** **94**
5.1 Newton-Cotes Methods . . . 95
5.2 Gaussian Quadrature . . . 97
5.3 Monte Carlo Integration . . . 100
5.4 Quasi-Monte Carlo Integration . . . 102
5.5 An Integration Toolbox . . . 104
5.6 Numerical Differentiation . . . 107
5.7 Initial Value Problems . . . 114
Exercises . . . 121
Bibliographic Notes . . . 126

**6 Function Approximation** **127**
6.1 Interpolation Principles . . . 128
6.2 Polynomial Interpolation . . . 131
6.3 Piecewise Polynomial Splines . . . 137
6.4 Piecewise-Linear Basis Functions . . . 143
6.5 Multidimensional Interpolation . . . 145
6.6 Choosing an Approximation Method . . . 149
6.7 An Approximation Toolkit . . . 150
6.8 Solving Functional Equations . . . 158
6.8.1 Cournot Oligopoly . . . 159
6.8.2 Function Inverses . . . 162
6.8.3 Boundary Value Problems . . . 164
Exercises . . . 172
Bibliographic Notes . . . 176

**7 Discrete State Models** **177**
7.1 Discrete Dynamic Programming . . . 178
7.2 Economic Examples . . . 180
7.2.1 Mine Management . . . 180

7.2.2 Asset Replacement - I . . . 181
7.2.3 Asset Replacement - II . . . 182
7.2.4 Option Pricing . . . 183
7.2.5 Job Search . . . 184
7.2.6 Optimal Irrigation . . . 185
7.2.7 Optimal Growth . . . 186
7.2.8 Renewable Resource Problem . . . 187
7.2.9 Bioeconomic Model . . . 188
7.3 Solution Algorithms . . . 189
7.4 Dynamic Simulation Analysis . . . 193
7.5 A Discrete Dynamic Programming Toolbox . . . 195
7.6 Numerical Examples . . . 198
7.6.1 Mine Management . . . 198
7.6.2 Asset Replacement - I . . . 201
7.6.3 Asset Replacement - II . . . 202
7.6.4 Option Pricing . . . 202
7.6.5 Job Search . . . 204
7.6.6 Optimal Irrigation . . . 206
7.6.7 Optimal Growth . . . 208
7.6.8 Renewable Resource Problem . . . 208
7.6.9 Bioeconomic Model . . . 208
Exercises . . . 210

**8 Continuous State Models: Theory 218**
8.1 Continuous State Dynamic Programming . . . 219
8.2 Continuous State Discrete Choice Models . . . 221
8.2.1 Asset Replacement . . . 221
8.2.2 Timber Cutting . . . 222
8.2.3 American Option Pricing . . . 222
8.2.4 Industry Entry and Exit . . . 223
8.2.5 Job Search . . . 224
8.3 Continuous State Continuous Choice Models . . . 225
8.3.1 Optimal Economic Growth . . . 227
8.3.2 Public Renewable Resource Management . . . 229
8.3.3 Private Nonrenewable Resource Management . . . 230
8.3.4 Optimal Water Management . . . 231
8.3.5 Optimal Monetary Policy . . . 233
8.3.6 Production-Adjustment Model . . . 234
8.3.7 Production-Inventory Model . . . 235
8.3.8 Optimal Feeding . . . 236

8.4 Linear-Quadratic Control . . . 237
8.5 Dynamic Games . . . 239
8.5.1 Capital-Production Game . . . 240
8.5.2 Risk-Sharing Game . . . 241
8.5.3 Marketing Board Game . . . 241
8.6 Rational Expectations Models . . . 242
8.6.1 Asset Pricing Model . . . 244
8.6.2 Competitive Storage . . . 245
8.6.3 Government Price Controls . . . 247
Exercises . . . 249

**9 Continuous State Models: Methods 257**
9.1 Traditional Solution Methods . . . 258
9.2 The Collocation Method . . . 260
9.3 Postoptimality Analysis . . . 269
9.4 Computational Examples . . . 271
9.4.1 Asset Replacement . . . 271
9.4.2 Timber Cutting . . . 274
9.4.3 Optimal Economic Growth . . . 276
9.4.4 Public Renewable Resource Management . . . 281
9.4.5 Private Nonrenewable Resource Management . . . 284
9.4.6 Optimal Monetary Policy . . . 288
9.4.7 Production-Adjustment Model . . . 292
9.5 Dynamic Game Methods . . . 296
9.5.1 Capital-Production Game . . . 301
9.5.2 Income Redistribution Game . . . 305
9.6 Rational Expectations Methods . . . 309
9.6.1 Asset Pricing Model . . . 313
9.6.2 Competitive Storage . . . 315
9.7 Comparison of Solution Methods . . . 320
Exercises . . . 322

**10 Continuous Time - Theory & Examples 330**
10.1 Arbitrage Based Asset Valuation . . . 330
10.2 Stochastic Control . . . 338
10.2.1 Boundary Conditions . . . 340
10.2.2 Choice of the Discount Rate . . . 342
10.2.3 Euler Equation Methods . . . 344
10.2.4 Examples . . . 345
10.3 Free Boundary Problems . . . 355

10.3.1 Impulse Control . . . 357
10.3.2 Barrier Control . . . 361
10.3.3 Discrete State/Control Problems . . . 363
10.3.4 Stochastic Bang-Bang Problems . . . 369
Exercises . . . 376
Appendix A: Dynamic Programming and Optimal Control Theory . . . 388
Appendix B: Deriving the Boundary Conditions for Resetting Problems . . 390
Appendix C: Deterministic Bang-Bang Problems . . . 392
Bibliographic Notes . . . 395

**11 Continuous Time - Methods** **397**
11.1 Solving Arbitrage-based Valuation Problems . . . 398
11.1.1 Extensions and Refinements . . . 401
11.2 Solving Stochastic Control Problems . . . 408
11.3 Free Boundary Problems . . . 419
11.3.1 Multiple Value Functions . . . 430
11.3.2 Finite Horizon Problems . . . 443
Exercises . . . 452
Bibliographic Notes . . . 461

**A Mathematical Background** **463**
A.1 Normed Linear Spaces . . . 463
A.2 Matrix Algebra . . . 466
A.3 Real Analysis . . . 468
A.4 Markov Chains . . . 470
A.5 Continuous Time Mathematics . . . 471
A.5.1 Ito Processes . . . 471
A.5.2 Forward and Backward Equations . . . 475
A.5.3 The Feynman-Kac Equation . . . 478
Bibliographic Notes . . . 480

**B A MATLAB Primer** **481**
B.1 The Basics . . . 481
B.2 Conditional Statements And Looping . . . 486
B.3 Scripts and Functions . . . 487
B.4 Debugging . . . 492
B.5 Other Data Types . . . 494
B.6 Programming Style . . . 495

**Web Resources** **497**

**References** **498**

**Index** **503**

# List of Tables

5.1. Errors for Selected Quadrature Methods . . . . . 98
5.2. Approximation Errors for Alternative Quasi-Monte Carlo Methods . . 103

6.1. Errors for Selected Interpolation Methods . . . . . 150

7.1 Optimal Labor Participation Rule . . . . . 206
7.2 Survival Probabilities . . . . . 211
7.3 Optimal Foraging Strategy . . . . . 211

9.1 Execution Times and Approximation Error for Selected Continuous-Space Approximation Methods . . . . . 323

10.1. Known Solutions to the Optimal Harvesting Problem . . . . . 346
10.2. Types of Free Boundary Problems . . . . . 355

11.1. Option Pricing Approximation Errors . . . . . 407

# List of Figures


3.1. demslv10 . . . . . 34
3.2. demslv11 . . . . . 35
3.3. demslv11 . . . . . 36
3.4. demslv12 . . . . . 39
3.5. demslv11 . . . . . 40
3.6. demslv12 . . . . . 43
3.7. demslv13 . . . . . 50
3.8. demslv14 . . . . . 52
3.9. demslv15 . . . . . 53
3.10. demslv16 . . . . . 55

4.1. demopt01 . . . . . 66
4.2. demopt02 . . . . . 67
4.3. demopt03 . . . . . 69
4.4. demopt04 . . . . . 74
4.5. demopt04 . . . . . 75
4.6. demopt05 . . . . . 76
4.7. demopt06 . . . . . 80
4.8. demopt06 . . . . . 81

5.1. demqua01 . . . . . 104
5.2. demqua01 . . . . . 105
5.3. demdif01 . . . . . 111
5.4. demdif02 . . . . . 112
5.5. demdif03 . . . . . 119

6.1. demapp01 . . . . . 132
6.2. demapp02 . . . . . 133
6.3. demapp01 . . . . . 134
6.4. demapp01 . . . . . 135
6.5. demapp01 . . . . . 139

6.6. demapp01 . . . . . 141
6.7. demapp04 . . . . . 142
6.8. demapp04 . . . . . 143
6.9. demapp05 . . . . . 152
6.10. demapp05 . . . . . 153
6.11. demapp05 . . . . . 154
6.12. demapp05 . . . . . 155
6.13. demapp06 . . . . . 157
6.14. demapp09 . . . . . 162
6.15. demapp09 . . . . . 163
6.16. demapp09 . . . . . 164
6.17. demapp10 . . . . . 165
6.18. demapp10 . . . . . 166
6.19. dembvp02 . . . . . 170
6.20. dembvp02 . . . . . 171

7.1. demddp01 . . . . . 200
7.2. demddp04 . . . . . 204
7.3. demddp06 . . . . . 208

9.1. demdp01 . . . . . 273
9.2. demdp02 . . . . . 276
9.3. demdp07 . . . . . 279
9.4. demdp08 . . . . . 283
9.5. demdp09 . . . . . 286
9.6. demdp11 . . . . . 291
9.7. demdp12 . . . . . 294
9.8. demgame01 . . . . . 305
9.9. demgame02 . . . . . 308
9.10. demrem01 . . . . . 315
9.11. demrem02 . . . . . 319

10.1. demfb05 . . . . . 374

11.1. demfin01 . . . . . 401
11.2. demfin01 . . . . . 402
11.3. demfin02 . . . . . 408
11.4. demsc1 . . . . . 415
11.5. demsc03 . . . . . 420
11.6. demsc03 . . . . . 420
11.7. demfb01 . . . . . 425

11.8. demfb02 . . . . . 429
11.9. demfb02 . . . . . 429
11.10. demfb03 . . . . . 433
11.11. demfb03 . . . . . 435
11.12. demfb03 . . . . . 437
11.13. demfb03 . . . . . 437
11.14. demfb04 . . . . . 442
11.15. demfb04 . . . . . 442
11.16. demfb04 . . . . . 443
11.17. demfin04 . . . . . 446
11.18. demfin04 . . . . . 446
11.19. demfin04 . . . . . 447
11.20. demfb05 . . . . . 451

# Glossary of Matlab Terms

A list of MATLAB terms used in this text. For a complete list, see MATLAB documentation.

| | |
|---|---|
| chol | computes the Cholesky decomposition of a symmetric positive definite matrix |
| diag | returns the diagonal elements of a matrix as a vector |
| disp | displays results on the screen |
| eps | machine precision (the largest number that, added to 1, returns 1) |
| eye | returns an order $n$ identity matrix: `I`n=eye(n) |
| feval | evaluates a function referred to by name: `f`eval(f,x,y) |
| find | produces an index of values meeting a stated condition: `f`ind([-1 3 -2 0]¿=0) returns `[`2 4] |
| inline | creates a function from a string that behaves like a function file: `f=inline('x.^2+2*y','x','y')` |
| inv | matrix inverse |
| length | the number of elements in a vector: `l`ength([0 5 2]) returns `3` |
| norm | vector or matrix norm (default is the 2-norm) |
| rand | produces random uniformly distributed values on [0,1]: `x`=rand(m,n) |
| randn | produces random standard normal (Gaussian) variates: `x`=randn(m,n) |
| realmax | the largest real number representable in MATLAB |
| reshape | changes the size of a matrix without changing the total number of elements: `r`eshape([1 1;1 1;2 2;2 2],2,4]) returns `[`1 2 1 2;1 2 1 2] |
| sum | sums the elements of a vector or columns of a matrix: `s`um([1 2 3] returns `6` |
| tril | zeros the above diagonal elements of a matrix: `t`ril([1 2;3 4]) returns `[`1 0;3 4] |
| triu | zeros the below diagonal elements of a matrix: `t`riu([1 2;3 4]) returns `[`1 2;0 4] |

# 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 approximate 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 closed form solutions. 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 a closed form 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 explicitly 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 explicitly soluble model.

Despite resistance by some, 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 addresses the difficulties typically encountered by economists attempting to learn and apply numerical methods in several ways. 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 financial, agricultural, resource and macro- economics. And third, we include with the textbook an extensive 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 six chapters, we develop basic numerical methods, including solving linear and nonlinear equation methods, complementarity methods, finite-dimensional optimization, numerical integration and differentiation, and function approximation. In these chapters, we develop appreciation for basic numerical techniques by illustrating their application to equilibrium and optimization models familiar to most economists. The last five chapters of the book are devoted to methods for solving 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.

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 toolbox of MATLAB utilities and demonstration programs to assist interested readers develop their own computational economic applications.

The CompEcon toolbox can be obtained via the internet at the URL:

http://?? All of the figures and tables in this book were generated by MATLAB demonstration files provided with the toolbox (see List of Tables and List of Figures for file names). Once the toolbox is installed, these can be run by typing the appropriate file name at the MATLAB command line. For those not familiar with the MATLAB programming language, a primer in provided in Appendix B.

The text contains many code fragments, which, in some cases, have been simplified for expositional clarity. This generally consists of eliminating the explicit setting of optional parameters and not displaying code that actually generates tabular or graphical output. The demonstration and function files provided in the toolbox contain fully functioning versions. In many cases the toolbox versions of functions described in the text have optional parameters that can be altered by the user user the toolbox function `optset`. The toolbox is described in detail in ?? on page ??.

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 sub-disciplines. It is our hope that this book will make accessible a range of computational tools that will enable economists to analyze 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 explicit answer using Algebra and Calculus. No 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:

$$a = 0.5 + 0.5E[p].$$

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

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

that is entirely sold at a market clearing price

$$p = 3 - 2q.$$

Assume the random yield $y$ is exogenous and distributed normally with a mean 1 and variance 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

$$p = 3 - 2(0.5 + 0.5E[p])\tilde{y}.$$

Taking expectations on both sides

$$E[p] = 3 - 2(0.5 + 0.5E[p]),$$

she would solve for the equilibrium expected price $E[p] = 1$. She would conclude that the equilibrium acreage is $a = 1$ and the equilibrium price distribution has a variance 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

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

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

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

As before, she takes expectations on both sides

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

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 $\max(E[p], 1)$. The resulting expression is easily solved for $E[p] = 1$. 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, which is discussed in Section 3.3 on page 35.

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 the CompEcon library routine `qnwnorm`, 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)
```

[1] The function `qnwnorm`, is discussed in Chapter 5.

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.

**Exercises**

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

1.2. 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 $x$?

1.3. 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 $t = 1960, 1961, \ldots, 2001$, assuming that the $\epsilon_t$ 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.

1.4. Consider the rational expectations commodity market model of discussed on page 2, 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?

# Chapter 2

# Computer Basics and Linear Equations

## 2.1 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 and avoid errors. 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, incorrectly, likely result in $x = 0$, whereas the second would result, correctly, in $x = 10^{-20}$. 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 it will suffice for our purposes to suppose that floating point numbers are stored in the form $m2^e$, where $m$ and $e$ are integers with $-2^b \leq m < 2^b$ and $-2^d \leq e < 2^d$. For example,

the number $-3210.48$ cannot be represented precisely as an ordinary double precision number but is approximately equal to $-7059920181484585\times 2^{-41}$. The value of the approximation, to 32 decimal digits, is equal to $-3210.4800000000000181898940354586$, implying that the error in representing $-3210.48$ is approximately $2^{-42}$.

Consider now what happens when arithmetic operations are performed. If $m_1 2^{e_1}$ is multiplied by $m_2 2^{e_2}$, the exact result is $m_1 m_2 2^{e_1+e_2}$. If $m_1 m_2$ is outside the range $[-2^b, 2^b)$, it will need to be divided by powers of 2 until it is within this range and the exponent will need to be adjusted accordingly. In the process of dividing $m_1 m_2$, any remainders will be lost. This means it is possible to perform the operation `(x*y)/y` and have the result not equal `x`; instead it may be off by 1 in its least significant digit. Furthermore, if $e_1 + e_2$ (plus any adjustment arising from the division) is greater than $2^d$ or less than $-2^d$, the result cannot be represented. This is a situation known as *overflow*. In MATLAB, overflow produces a result that is set to `inf` or `-inf`. Further operations may be possible and produce sensible results, but, more often than not, the end result of overflow is useless.

Addition is also problematic. Suppose $e_1 > e_2$; then

$$m_1 2^{e_1} + m_1 2^{e_2} = \left(m_1 + \frac{m_2}{2^{e_1-e_2}}\right) 2^{e_1}.$$

The computer, however, will truncate $m_2/2^{e_1-e_2}$, so the result will not be exact. It is therefore possible to perform `x+y`, for $y \neq 0$ and have the result equal $x$; this will occur if $m_2 < 2^{e_1-e_2}$. Although odd, the result is nonetheless accurate to its least significant digit.

Of all the operations on floating point numbers, the most troublesome is subtraction, particularly when a large number is subtracted from another large number. Consider, for example, what happens when one performs `1000000.2-1000000.1`. The result, of course, should equal 0.1 but instead equals (on Pentium processor) 0.09999999997672. The reason for this strange behavior is that the two number being operated on cannot be represented exactly. On a Pentium processor, the floating points numbers used are actually

$$8589935450993459 \times 2^{-33} = 1000000.09999999997671693563461 30$$

and

$$8589936309986918 \times 2^{-33} = 1000000.19999999995343387126922 61.$$

The result obtained from subtracting the first from the second is therefore

$$(8589936309986918 - 8589935450993459)2^{-33} = 0.09999999997672$$

as we found above. The error is approximately $-2.3283 \times 10^{-11}$, which is roughly the same order of magnitude as $2^{-34}$.

Although one's first impression may be to minimize the importance of finite precision arithmetic, serious problems can arise if one is not careful. Furthermore, these problems may result in strange behavior that is hard to track down or erroneous results that may, or may not, be detected.

Consider, for example, the computation of the function

$$\phi^-(y,z) = y + z - \sqrt{y^2 + z^2}.$$

This function is used in solving complementarity problems and is discussed in Section 3.7 on page 53. Most of the time it can be computed as written and no problems will arise. When one of the values gets large relative to the other, however, the obvious way of coding can fail due to overflow or, worse, can produce an incorrect answer.

Suppose that $|y| > |z|$. One problem that can arise is that $y$ is so big that $y^2$ overflows. The largest real number representable on a machine can be found with the MATLAB command `realmax` (it is approximately $2^{1024} \approx 10^{308}$ for most double precision environments). Although this kind of overflow may not happen often, it could have unfortunate consequences and cause problems that are hard to detect.

Even when $y$ is not that big, if it is big relative to $z$, several problems can arise. The first of these is easily dealt with. Suppose we evaluate

$$y + z - \sqrt{y^2 + z^2}$$

when $|y|$ is large enough so $y + z$ is evaluated as $y$. This implies that $\sqrt{y^2 + z^2}$ will be evaluated as $|y|$. When $y < 0$, the expression is evaluated as $2y$, which is correct to the most significant digit. When $y > 0$, however, we get 0, which may be very far from correct. If the expression is evaluated in the order

$$y - \sqrt{y^2 + z^2} + z$$

the result will be $z$, which is much closer to the correct answer.

An even better approach is to use

$$\phi^-(y,z) = y\Big(1 - \text{sign}(y)\sqrt{1 + \epsilon^2} + \epsilon\Big),$$

where $\epsilon = z/y$. Although this is algebraically equivalent, it has very different properties. First notice that the chance of overflow is greatly reduced because $1 \le 1+\epsilon^2 \le 2$ and so the expression in ( ) is bounded on $[\epsilon, 4]$. If $1 + \epsilon^2$ is evaluated as 1 (i.e., if $\epsilon$ is less than the square root of machine precision), this expression yields $2y$ if $y < 0$ and $y\epsilon = z$ if $y > 0$.

This is a lot better, but one further problem arises when $y > 0$ with $|y| \gg |z|$. In this case there is a cancellation due to the expression of the form

$$z = 1 - \sqrt{1 + \epsilon^2}$$

The obvious way of computing this term will result in loss of precision as $\epsilon$ gets small. Another expression for $z$ is

$$z = -\frac{\left(1-\sqrt{1+\epsilon^2}\right)^2+\epsilon^2}{2\sqrt{1+\epsilon^2}}.$$

Although this is more complicated, it is accurate regardless of the size of $\epsilon$. As $\epsilon$ gets small, this expression will be approximately $\epsilon^2/2$. Thus, if $\epsilon$ is about the size of the square root of machine precision ($2^{-26}$ on most double precision implementations), $z$ would be computed to machine precision with the second expression, but would be computed to be 0 using the first, i.e., no significant digits would be correct.

Putting all of this together, a good approach to computing $\phi^-(y,z)$ when $|y| \geq |z|$ uses

$$\phi^-(y,z) = \begin{cases} y(1+\sqrt{1+\epsilon^2}+\epsilon) & \text{if } y < 0 \\ y\left(\epsilon - \frac{\left(1-\sqrt{1+\epsilon^2}\right)^2+\epsilon^2}{2\sqrt{1+\epsilon^2}}\right) & \text{if } y > 0 \end{cases}$$

where $\epsilon = z/y$ (reverse $z$ and $y$ if $|y| < |z|$).

MATLAB has a number of special numerical representations relevant to this discussion. We have already mentioned `inf` and `-inf`. These arise not only from overflow but from division by 0. The number `realmax` is the largest floating point number that can be represented; `realmin` is the smallest positive (normalized) number representable.[1] In addition, `eps` represents the machine precision, defined as the first number greater than 1 that can be represented as a floating point number. Another way to say this is, for any $0 \leq \epsilon \leq$ `eps`$/2$, $1+\epsilon$ will be evaluated as 1 (i.e., `eps` is equal to $2^{1-b}$).[2] All three of these special values are hardware specific.

In addition, floating point numbers may get set to `NaN`, which stands for not-a-number. This typically results from a mathematically undefined operation, such as `inf-inf` and `0/0`. It does not result, however, from `inf/0`, `0/inf` or `inf*inf` (these result in `inf`, 0 and `inf`). Any arithmetic operation involving a `NaN` results in a `NaN`.

Roundoff error is only one of the pitfalls in evaluating mathematical expressions. 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 operations in evaluating expressions. Essentially this list includes the four arithmetic operations of addition, subtraction, multiplication and division, as well as logical operations of comparison. Other common functions,

[1] A denormalized number is one that non-zero, but has an exponent equal to its smallest possible value.

[2] $2^0 + 2^{-b} = (2^b+1)2^{-b}$ cannot be represented and must be truncated to $(2^{b-1})2^{1-b} = 1$. $2^0 + 2^{1-b} = (2^{b-1}+1)2^{1-b}$, on the other hand, can be represented.

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.[3]

For nonstandard problems, we must often rely on our own abilities as numerical analysts (or know when to seek help). Being aware of some of the pitfalls should help us avoid them.

## 2.2 Data Storage

MATLAB's basic data type is the matrix, with a scalar just a $1 \times 1$ matrix and an n-vector an $n \times 1$ or $1 \times n$ matrix. MATLAB keeps track of matrix size by storing row and column information about the matrix along with the values of the matrix itself. This is a significant advantage over writing in low level languages 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 $m \times n$ 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 supports a sparse matrix data type, which efficiently keeps track of only the

[3]Incidently, the Taylor series representation of the exponential function does not result in an efficient computational algorithm.

non-zero elements of the original matrix and their locations. In this storage scheme, the non-zero entries and the row indices are stored in two vectors of the same size. 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 $j$th element of the column indicator vector to find where the $j$th column starts and then searches the row indicator vector for the $i$th 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 storage location $(i-1)m+j$. Accessing 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.

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.

Fortunately, for many applications, users don't even need to be aware of whether matrices are stored in sparse or full form. MATLAB is designed so most functions work with any mix of sparse or full representations. Furthermore, sparsity propagates in a reasonably intelligent fashion. For example, a sparse times a full matrix or a spare plus a full matrix results in a full matrix, but if a sparse and a full matrix are multiplied element-by-element (using the ".*" operator) a sparse matrix results.

## 2.3 Linear Equations and the L-U Factorization

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.

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*:

$$x_1 = b_1/a_{11}$$

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

This clearly works only if all of the diagonal elements are non-zero (i.e., if the matrix is nonsingular). The algorithm can 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; by interchanging the second and third rows, a lower diagonal matrix results. The matrix $U$ is upper triangular. Solving $Ly = b$ for $y$ using forward substitution involves first solving for $y_1$, then for $y_3$, and finally for $y_2$. Given the solution $y = [10 \quad 7 \quad -2]^\top$, the linear equation $Ux = y$ can the be solved using backward substitution, yielding the solution of the original linear equation, $x = [-1 \quad 2 \quad 1]^\top$.

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 $10 \times 10$ 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 (see Section 2.7 on page 22).

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 $mn^3/3 + mn^2$ 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.

## 2.4 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 the 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 columns. 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.

## 2.5 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

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

where $M$ is a large positive number.

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

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

In theory, applying forward and backward substitution yields the solution $x_1 = M/(M+1)$ and $x_2 = (M+2)/(M+1)$, 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 $M$ on a computer simply returns the value $M$. Thus, in the first step of the backward substitution, $x_2$ is computed, not as $(M+2)/(M+1)$, but rather as $M/M$, which is exactly one. Then, in the second step of backward substitution, $x_1 = -M(1-x_2)$ 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 $-M^{-1}$ 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:

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

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

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

Backward and forward substitution yield the theoretical results $x_1 = 1 - M^{-1}$ and $x_2 = M^{-1} + 1 + M^{-1}(1 - M^{-1})$. 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.

## 2.6 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 $\delta b$ in the data vector $b$ induces a large change $\delta x$ 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 decimals 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 we will encounter again in Chapter 6.

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.

## 2.7 Special Linear Equations

Gaussian elimination can be accelerated for matrices possessing certain special structures. Two such classes arising frequently in computational economic analysis 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

$$A = U^\top U$$

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

$$Ax = U^\top Ux = U^\top (Ux) = b$$

may be solved efficiently by using forward substitution to solve

$$U^\top y = b$$

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 operations with zeros.

## 2.8 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*, $\lambda$. 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 $\lambda$ 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**

2.1. It is well known that a quadratic equation

$$ax^2 + bx + c = 0$$

has two roots given by

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

There are, however, other mathematically correct ways of expressing the quadratic equation. For example, it could be written as

$$\frac{-2c}{b \pm \sqrt{b^2 - 4ac}}$$

or, indeed, as either

$$\frac{-b}{2a}(1 \pm \sqrt{1 - 4ac/b^2}).$$

or

$$\frac{-2c}{b(1 \pm \sqrt{1 - 4ac/b^2})}$$

(you can derive these by noting that $4ac = (b + \sqrt{b^2 - 4ac})(b - \sqrt{b^2 - 4ac})$. Discuss the relative merits of these alternative ways of computing the roots. Under what circumstances will each produce inaccurate results. Based on these considerations, write a MATLAB function that accepts $a$, $b$ and $c$ and returns the two roots.

2.2. 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?

2.3. 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`.

2.4. 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 (Appendix A, 466) and the result that $||My|| \leq ||M||\ ||y||$ for any square matrix $M$ and conformable vector $y$.

## Bibliographic Notes

Good introductory discussions of computer basics are contained in Gill et al., Press et al. and Kennedy and Gentle. These references also all contain discussions of computational aspects of linear algebra and matrix factorizations. A standard in depth treatment of computational linear algebra is Golub and van Loan. Most textbook on linear algebra also include discussions of Gaussian elimination and other factorizations; see, for example, Leon.

We have only discussed the two matrix factorizations that are most important for the remainder of this text. A number of other factorizations exist and have uses in computational economic analysis, making them worth mentioning briefly (see refenereces cited above for more details).

The first is the eigenvalue/eigenvector factorization. Given $A$ $(n \times n)$, this finds $n \times n$ matrices $Z$ and $D$, with $D$ diagonal, that satisfy $AZ = ZD$. The columns of $Z$ and the diagonal elements of $D$ form eigenvector, eigenvalue pairs. If $Z$ is nonsingular, this leads to a factorization of the form $A = ZDZ^{-1}$. It is possible, however, that $Z$ is singular (even if $A$ is not); such matrices are called defective. The eigenvalue/eigenvector factorization is unique (up to rearrangement and possible linear combinations of columns of $Z$ associated with repeated eigenvalues).

In general, both $Z$ and $D$ may be complex-valued, even if $A$ is real-valued. Complex eigenvalues arise in economic models that display cyclic behavior. In the special case that $A$ is real-valued and symmetric, the eigenvector matrix is not only guaranteed to be nonsingular but is orthonormal (i.e., $Z^\top Z = I$), so $A = ZDZ^\top$ and $Z$ and $D$ are real-valued.

Another factorization is the $QR$ decomposition, which finds a representation $A = QR$, where $Q$ is orthonormal and $R$ is triangular. This factorization is not unique; there are a number of algorithms that produce different values of $Q$ and $R$, including Householder and Givens transformations. $A$ need not be square to apply the QR decomposition.

Finally, we mention the singular-value decomposition (SVD), which finds $U$, $D$ and $V$, with $U$ and $V$ orthonormal and $D$ diagonal, that satisfies $A = UDV^\top$. The diagonal elements of $D$ are known as the *singular values* of $A$ and are nonnegative and generally order highest to lowest. In the case of a square, symmetric $A$, this is identical to the eigenvalue/eigenvector decomposition. The SVD can be used with non-square matrices.

The SVD is the method of choice for determining matrix condition and rank. The condition number is the ratio of the highest to the lowest singular value; the rank is the number of non-zero singular values. In practice, one would treat a singular value $D_{jj}$ as zero if $D_{jj} < \max_i(D_{ii})\epsilon$, for some specified value of $\epsilon$ (MATLAB sets $\epsilon$ equal to the value of the machine precision `eps` times the maximum of the number of rows

and columns of $A$).

We have only touched on iterative methods. These are mainly useful when solving large sparse systems that cannot be stored directly. See Golub and Ortega, Section 9.3, for further details and references.

Numerous software libraries that perform basic linear algebra computations are available, including LINPACK, LAPACK, IMSL and NAG. See Notes on Web Resources (page 497).

# 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 nonlinear equations. Nonlinear equations generally arise in one of two forms. In the nonlinear *rootfinding problem*, a function $f$ mapping $\Re^n$ 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

$$x = g(x).$$

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

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

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 $\epsilon$ in no more than in $\log((b-a)/\epsilon)/\log(2)$ 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 $1.5 \times 10^{-8}$, or eight decimal places. The sequence of iterates is illustrated in Figure 3.1. The function `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.

## 3.2 Function Iteration

*Function iteration* is a relatively simple technique that may be used to compute a fixed-point, $x = g(x)$, 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 $x = x - f(x)$.

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"

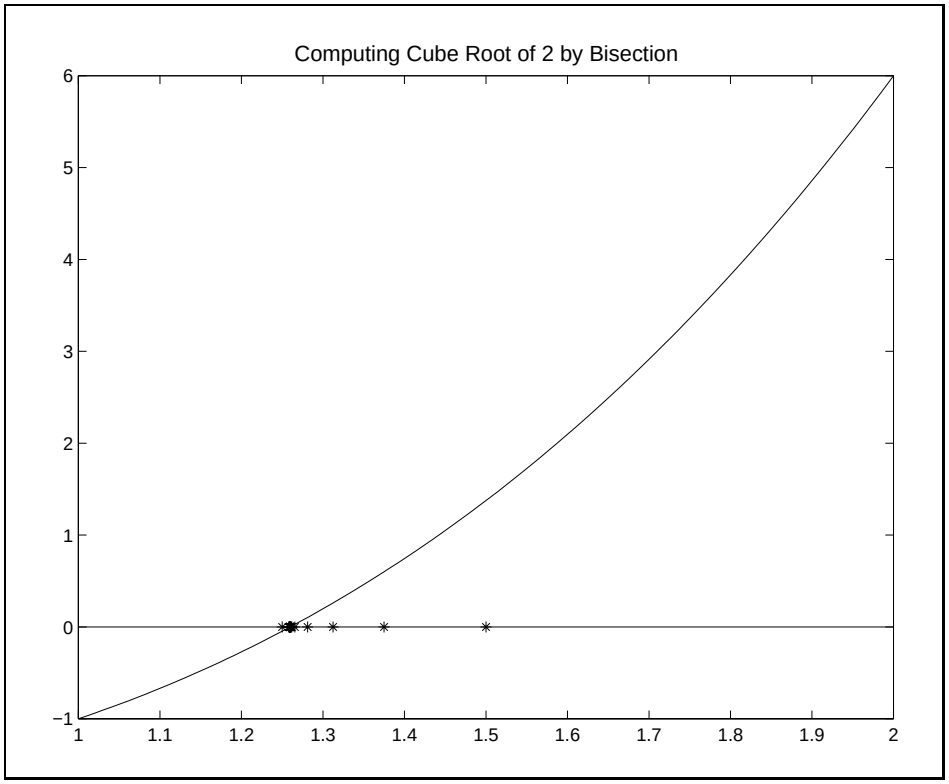


Figure 3.1

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

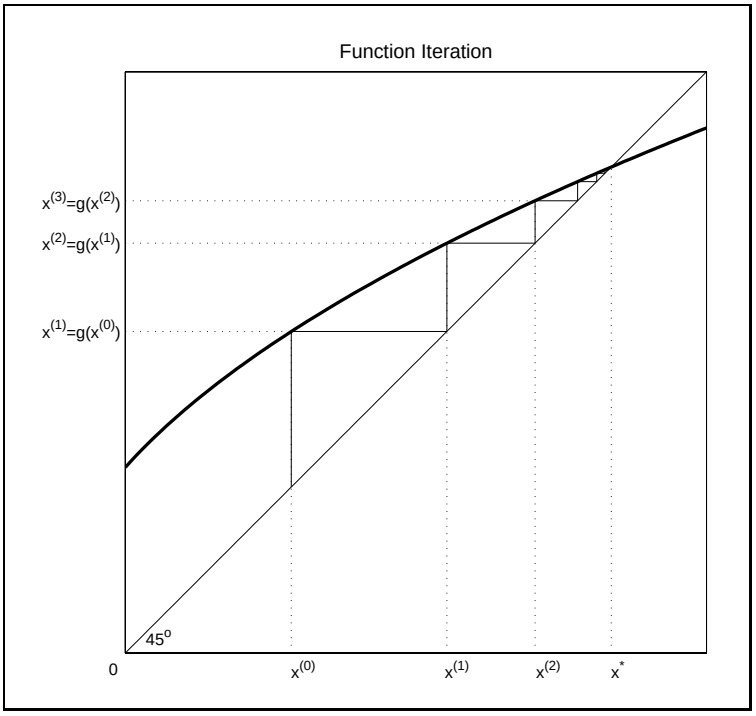


Figure 3.2

following script computes the fixed point $x^* = 1$ of $g(x) = x^{0.5}$ to a default tolerance of $1.5 \times 10^{-8}$ 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 $x = g(x)$ by recasting it as the rootfinding problem $f(x) = x - g(x) = 0$.

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.

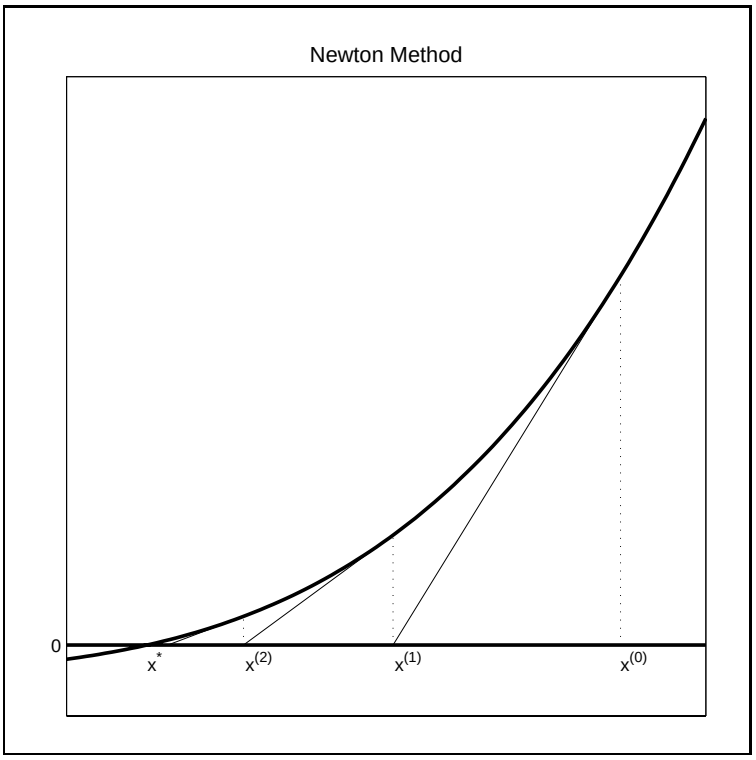


Figure 3.3

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 $x^{(k)}$:

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

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. The user inputs the name of the function file that computes $f$, a starting vector and any additional parameters to be passed to $f$ (the first input to $f$ must be $x$). The function has default values for the convergence tolerance and the maximum number of steps to attempt. These may be altered using `optset`.

**Example: Cournot Duopoly**
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('cournot',[0.2;0.2]);
```

will compute the equilibrium quantities $q_1 = 0.8396$ and $q_2 = 0.6888$ to the default tolerance of $1.5 \times 10^{-8}$. 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.

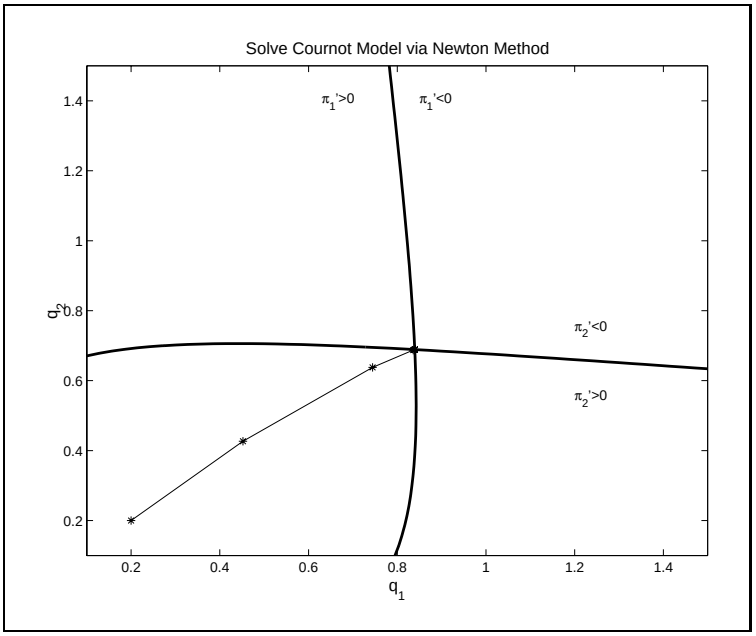


Figure 3.4

## 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:

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

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.

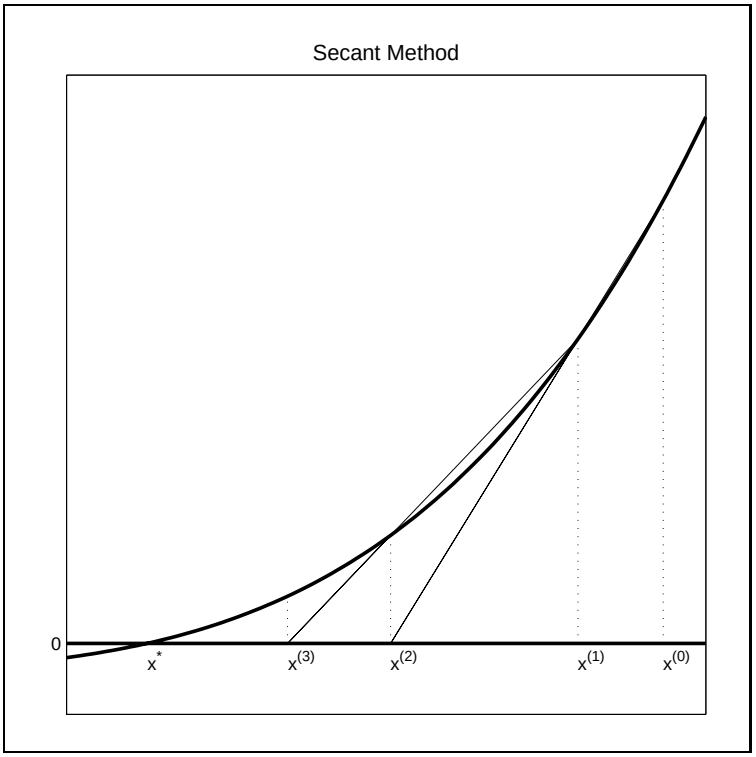


Figure 3.5

*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 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

[1]Numerical differentiation is discussed in Section 5.6 (page 107).

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

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

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:

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

This yields the iteration rule

$$A^{(k+1)} \leftarrow A^{(k)} + \left(f(x^{(k+1)}) - f(x^{(k)}) - A^{(k)} d^{(k)}\right) \frac{d^{(k)\top}}{d^{(k)\top} d^{(k)}}$$

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

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 iteration rule

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

and inverse update rule[2]

$$B^{(k+1)} \leftarrow B^{(k)} + \left((d^{(k)} - u^{(k)}) d^{(k)\top} B^{(k)}\right) / (d^{(k)\top} u^{(k)})$$

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

[2]This is a straightforward application of the Sherman-Morrison formula:

$$(A + uv^\top)^{-1} = A^{-1} + \frac{1}{1 + u^\top A^{-1} v} A^{-1} u v^\top A^{-1}.$$

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 Chapter 5 (page 107).

```
fjacinv = inv(fdjac(f,x));
fval = f(x);
for it=1:maxit
  fnorm = norm(fval);
  if fnorm<tol, break; end
  d = -(fjacinv*fval);
  x = x+d;
  fold = fval;
  fval = f(x);
  u = fjacinv*(fval-fold);
  fjacinv = fjacinv + ((d-u)*d'*fjacinv)/(d'*u);
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 function `cournot` listed on page 38 could be passed to `broyden`, with an initial guess of, say $q_1 = q_2 = 0.2$:

```
q = broyden('cournot',[0.2;0.2]);
```

yielding the equilibrium quantities $q_1 = 0.8396$ and $q_2 = 0.6888$ to the default tolerance of $1.5 \times 10^{-8}$. Note that the function `cournot` need not return the Jacobian of $f$ because the Broyden method does not require it. 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.

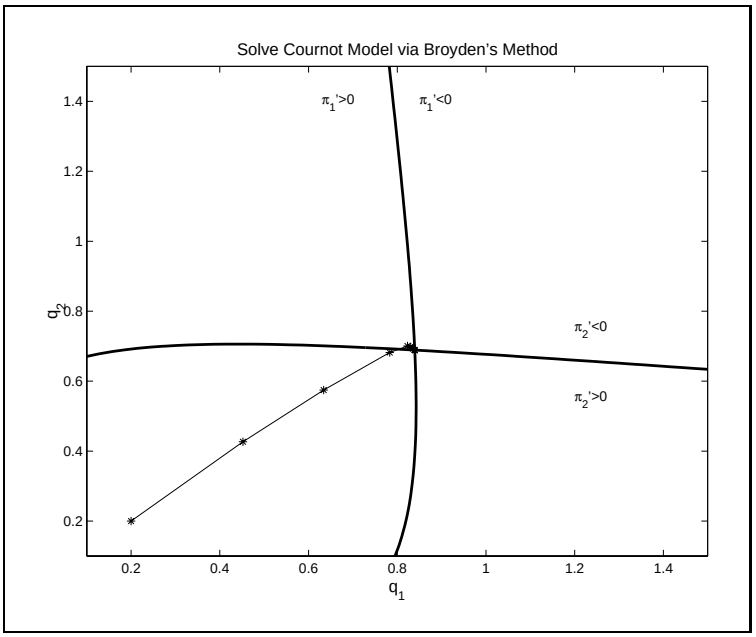


Figure 3.6

## 3.5 Problems With Newton Methods

Several difficulties commonly arise in the application of Newton and quasi-Newton methods to solving multivariate non-linear equations. 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. These problems can often be mitigated by appropriate action, though they cannot always be eliminated altogether.

The 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 $x + d$ does not offer an improvement over the current iterate $x$, then one "backsteps" toward the current iterate $x$ by repeatedly cutting $d$ in half until $x+d$ does offer an improvement. Whether a step $d$ 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 $\|f(x)\| > \|f(x + d)\|$. Backstepping prevents Newton and quasi-Newton methods from taking a large step in the wrong direction, substantially improving their robustness.

A simple backstepping algorithm will not necessarily prevent Newton type methods from getting stuck at a local minimum of $\|f(x)\|$. If $\|f(x)\|$ must decrease with each step, it may be difficult to find a step length that moves away from the current value of $x$. Most good root-finding algorithms employ so mechanism for getting unstuck. We use a very simple one in which the backsteps continue until either $\|f(x)\| > \|f(x + d)\|$ or $\|f(x + d/2)\| > \|f(x + d)\|$.

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
  d = -(fjac\fval);
  fnormold = inf;
  for backstep=1:maxsteps
     fvalnew = f(x+d);
     fnormnew = norm(fvalnew);
     if fnormnew<fnorm, break, end
     if fnormold<fnormnew, d=2*d; break, end
     fnormold = fnormnew;
     d = d/2;
  end
  x = x+d;
end
```

Safeguarded backstepping may also implemented with Broyden's method; 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 convergence 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 at $x$ and $x$ solves the complementarity problem $\mathrm{CP}(f, a, b)$, then $x$ solves the bounded maximization problem (see Section 4.6, page 79).

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.

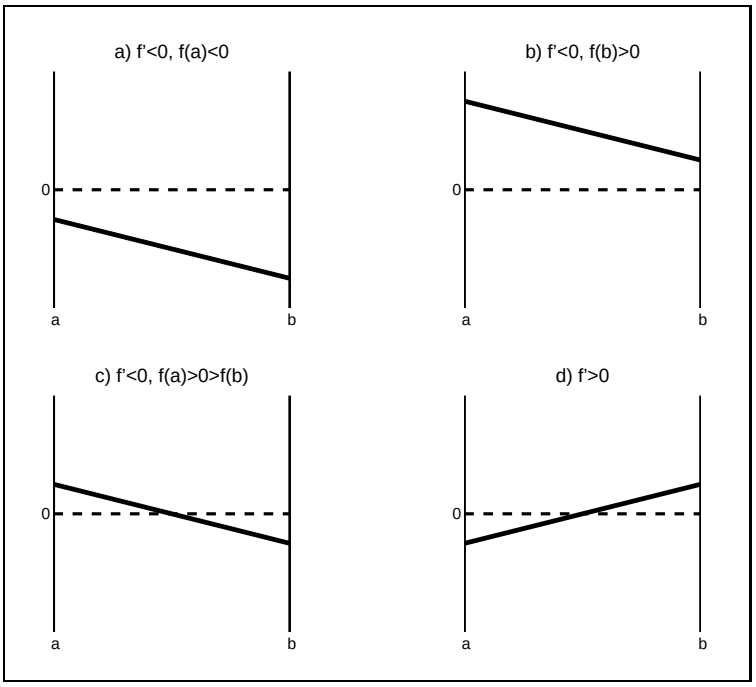


Figure 3.7

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 $(x-y)^\top(f(x)-f(y)) < 0$ 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 3.8d, $f$ 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 $f$:

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

Here, $I_{i\cdot}$ is the $i^{th}$ row of the identity matrix.

The following MATLAB script computes the solution of the complementarity problem $\mathrm{CP}(f, a, b)$ by applying Newton's method to the equivalent minmax rootfinding formulation. The script assumes that the user has provided the lower and upper

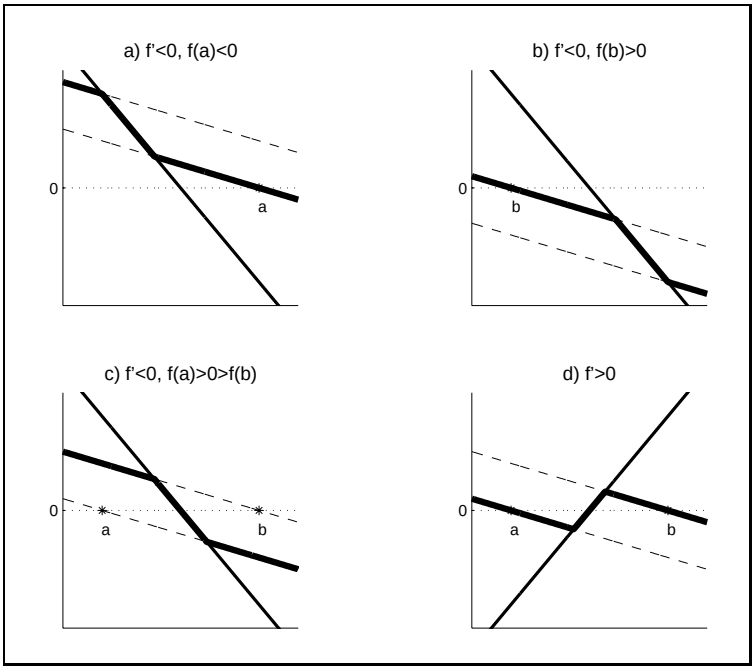


Figure 3.8

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 and possibly even causing it to cycle. 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[3] function

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

where

$$\phi_i^{\pm}(u, v) = u_i + v_i \pm \sqrt{u_i^2 + v_i^2}.$$

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 $\hat{f}$.

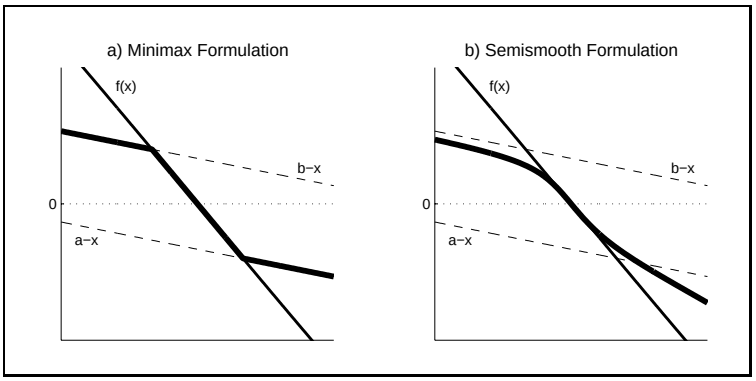


Figure 3.9

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 semismooth formulation one writes the MATLAB script `x=ncpsolve('f',a,b,x)`; to solve the complementarity problem using the minmax formulation one must change the default option using the MATLAB script `optset('ncpsolve','type','minmax')` before executing the `x=ncpsolve('f',a,b,x)` script.

[3] One could also use

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

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 $\hat{f}$ (dashed) and $\tilde{f}$ (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 $\hat{f}$ because 0 poses an upper bound on the positive Newton step. With $\tilde{f}$, 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 = ncpsolve('f',0,inf,0);
```

(this uses $x = 0$ as a starting value). To solve the complementarity problem using the minmax formulation, one executes the MATLAB script

```
optset('ncpsolve','type','minmax')
x = ncpsolve('f',0,inf,0);
```

In this example, the semismooth formulation will successfully 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

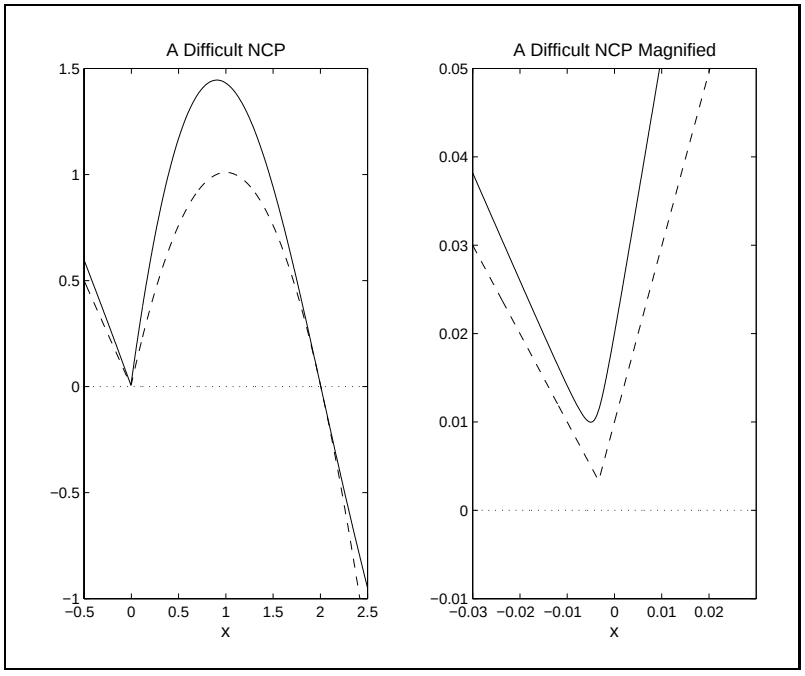


Figure 3.10

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**

3.1. The bisection method can fail if the initial interval doesn't bracket a root. Develop and implement in MATLAB a strategy that finds a root-bracketing interval.

3.2. If $x = \sqrt{c}$ then $x^2 - c = 0$.

a) Use this root condition to construct a Newton's method for determining the square root that uses only simple arithmetic operations (addition, subtraction, multiplication and division).

b) Given an arbitrary value of $c > 0$, how would you find a starting value to begin Newton's method?

c) Write a MATLAB procedure

```
function x=newtroot(c)
```

that implements the method. The procedure should be self-contained (i.e., it should not call a generic root-finding algorithm).

3.3. The computation of $\sqrt{1+c^2} - 1$ can fail due to overflow or underflow: when $c$ is large, squaring it can exceed the largest representable number (`realmax` in MATLAB), whereas when $c$ is small, the addition $1 + c^2$ will be truncated to 1.

Noting that $x = \sqrt{1+c^2} - 1$ is equivalent to the condition

$$(x+1)^2 - (1+c^2) = 0.$$

Determine the iterations of the Newton method for finding $x$ and a good starting value for the iterations. Write a MATLAB program that accepts $c$ and returns $x$, using only simple arithmetic operations (i.e., do not use power, log, square root operators). The procedure should be self-contained (i.e., it should not call a generic root-finding algorithm). Be sure to deal with the overflow problem.

3.4. Black-Scholes Option Pricing Formula

The Black-Scholes option pricing formula expresses the value of an option as a function of the current value of the underlying asset, $S$, the option's strike price $K$, the time-to-maturity on the option, $\tau$, the current risk-free interest rate, $r$, a dividend rate, $\delta$, and the volatility of the the price of the underlying asset, $\sigma$.

The formula for a call option is[4]

$$V(S,K,\tau,r,\delta,\sigma) = e^{-\delta\tau}S\Phi\left(d\right) - e^{-r\tau}K\Phi\left(d-\sigma\sqrt{\tau}\right)$$

where

$$d = \frac{\ln(e^{-\delta\tau}S) - \ln(e^{-r\tau}K)}{\sigma\sqrt{\tau}} + \tfrac{1}{2}\sigma\sqrt{\tau},$$

and $\Phi$ is the standard normal CDF:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-\frac{1}{2}z^2}dz.$$

a) Write a MATLAB procedure that takes the 6 inputs and returns the Black-Scholes option value:

```
V=BSVal(S,K,tau,r,delta,sigma)
```

The function `cdfn` provided in the `COMPECON` toolbox can be used to compute the standard normal CDF.

b) All of the inputs to the Black-Scholes formula are readily observable except $\sigma$. Market participants often want to determine the value of $\sigma$ implied by the market price of an option. Write a stand-alone that computes the so-called "implied volatility". The function should have the following calling syntax

```
sigma=ImpVol(S,K,tau,r,delta,V)
```

The algorithm should use Newton's method to solve (for $\sigma$) the root-finding problem $V - $`BSVal`$(S,K,\tau,r,\delta,\sigma)$. To do this you will need to use the derivative of the Black-Scholes formula with respect to $\sigma$, which can be shown to equal

$$\frac{\partial V}{\partial \sigma} = Se^{-\delta\tau}\sqrt{\tau/(2\pi)}e^{-0.5d^2}.$$

The program should be stand-alone, hence it should not call any root-finding solver such as `newton` or `broyden` or a numerical derivative algorithm. It may, however, call `BSVal` from part (a).

c) If the procedures you wrote for the previous two exercises are not vectorized, make them so. They should be able to accept a set of conformable matrices as inputs and return an appropriately sized result.

[4]This is known as the extended Black-Scholes formula because it includes the parameter $\delta$ not found in the original formula. The inclusion of $\delta$ generalizes the formula: for options on stocks $\delta$ represents a continuous percentage dividend flow, for options on currencies it is set to the interest rate in the foreign country and for options on futures it is set to $r$.

3.5. It was claimed (page 41) that the Broyden method chooses the approximate Jacobian to minimize a matrix norm subject to a constaint. Specifically

$$A^* \leftarrow A + (g - Ad)\frac{d^\top}{d^\top d}$$

with $g = f(x^{(k+1)}) - f(x^{(k)})$ and $d = x^{(k+1)} - x^{(k)}$, solves the problem

$$\min_{A^*} \sum_i \sum_j \left(A^*_{ij} - A_{ij}\right)^2.$$

subject to

$$g = A^* d.$$

Provide a proof of this claim.

3.6. 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.

3.7. A common problem in computation is finding the inverse of a cumulative distribution function (CDF). A CDF is a function, $F$, that is nondecreasing over some domain $[a, b]$ and for which $F(a) = 0$ and $F(b) = 1$. Write a function that uses Newton's method to solve inverse CDF problems. The function should take the following form:

```
x=icdf(p,F,x0,varargin)
```

where `p` is a probability value (a real number on [0,1]), `F` is the name of a MATLAB function file, and `x0` is a starting value for the Newton iterations.

The function file should have the form:

```
[F,f]=cdf(x,additional parameters)
```

For example, the normal CDF with mean $\mu$ and standard deviation $\sigma$ would be written:

```
function [F,f]=cdfnormal(x,mu,sigma)
  z=(x-mu)./sigma;
  F=cdfnorm(z);
  f=exp(-0.5*z.^2)./(sqrt(2*pi)*sigma);
```

You can test your code with the statement:

```
x-icdf(cdfnormal(x,0,1),'cdfnormal',0,0,1)
```

which should return a number close to 0.

3.8. 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 |

3.9. 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 interperiod 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 $\delta = 0.95$ 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 $s = 3$.

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

3.11. 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.

# Bibliographic Notes

Rootfinding problems have been studied for centuries (Newton's method bears its name for a reason). They are discussed in most standard references on numerical analysis. In depth treatments can be found in Dennis and Schnabel and in Ortega and Rheinboldt. Press et al. provides a discussion, with computer code, of both Newton's and Broyden's method and of backstepping.

Standard references on complementarity problems include Balinski and Cottle, Cottle et al. (1980), Cottle et al. (1992) and Ferris. Ferris and Pang provides an overview of applications of CPs.

We have broken with standard expositions of complementarity problems; the CP problem is generally stated to be

$$f(x) \geq 0,\ x \geq 0 \text{ and } x^\top f(x) = 0.$$

This imposes only a one-sided bound on $x$ at 0. Doubly bounded problems are often called mixed complementarity problems (MCPs) and are typically formulated as solving

$$\max(\min(f(x), x - a), x - b) = 0$$

rather than

$$\min(\max(f(x), a - x), b - x) = 0,$$

as we have done. If standard software for MCPs is used, the sign of $f$ should be reversed.

A number of approaches exist for solving CPs other than reformulation as a rootfinding problem. A well-studied and robust algorithm based on successive linearization is incorporated in the PATH algorithm described by Ferris et al., and Ferris and Munson. The linear complementarity problem (LCP) has received considerable attention and forms the underpinning for methods based on successive linearization. Lemke's method is perhaps the most widely used and robust LCP solver. It is described in the standard works cited above. Recent work on LCPs includes Kremers and Talman.

We have not discussed homotropy methods for solving nonlinear equations, but these may be desirable to explore, especially if good initial values are hard to guess. Judd, chapter 5, contains a good introduction, with economic applications and references for further study.

# 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 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.

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

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 Wierstrass Theorem, 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 $\epsilon$-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 $\epsilon$-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]

## 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 (or the local maximum is at one of the original endpoints).

[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.

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 $\alpha_2$ 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 first defines a MATLAB function that returns the value of the objective function at an arbitrary point. One then passes the name of this function, along with the lower and upper bounds for the search interval, to `golden`. 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.

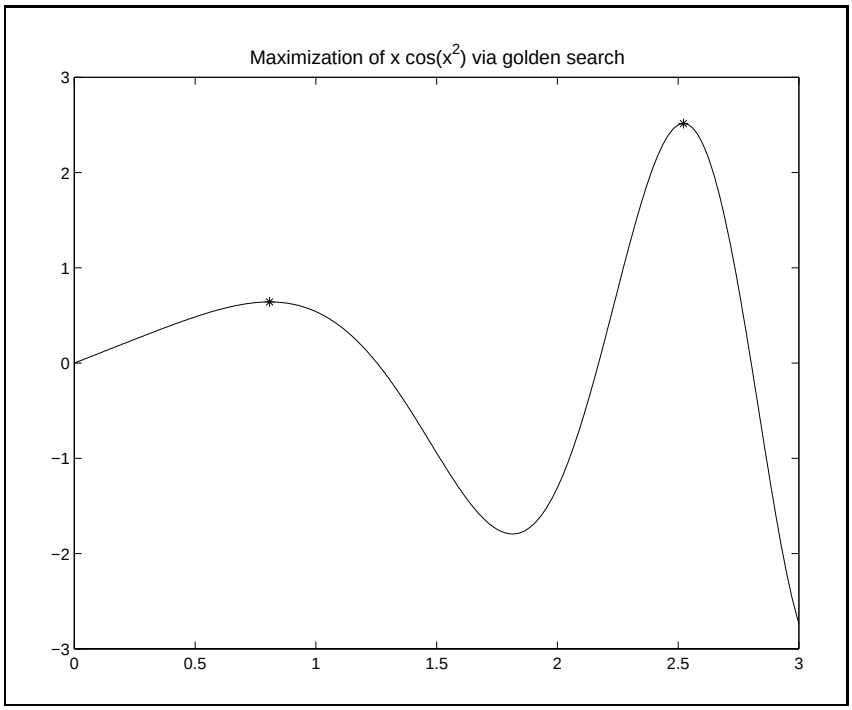


Figure 4.1

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 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).

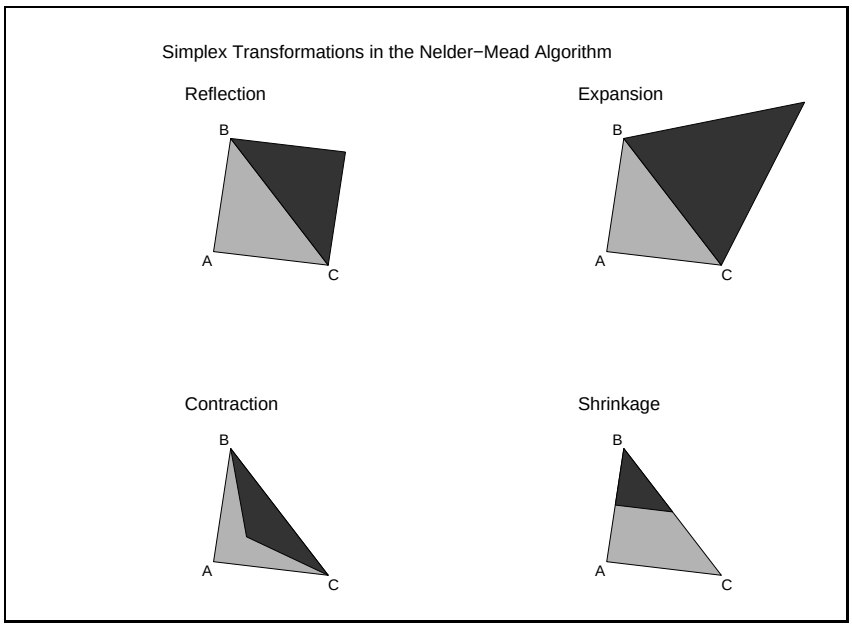


Figure 4.2

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 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 `neldermead` that maximizes a multivariate function using the Nelder-Meade method. To apply this function, one must first define a MATLAB function `f` that returns the value of the objective functions at an arbitrary point and then pass the name of this function along with a starting value `x` to `neldermeade`. Consider, for example, maximizing the "banana" function $f(x) = -100(x_2 - x_1^2)^2 - (1 - x_1)^2$, 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.

## 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.

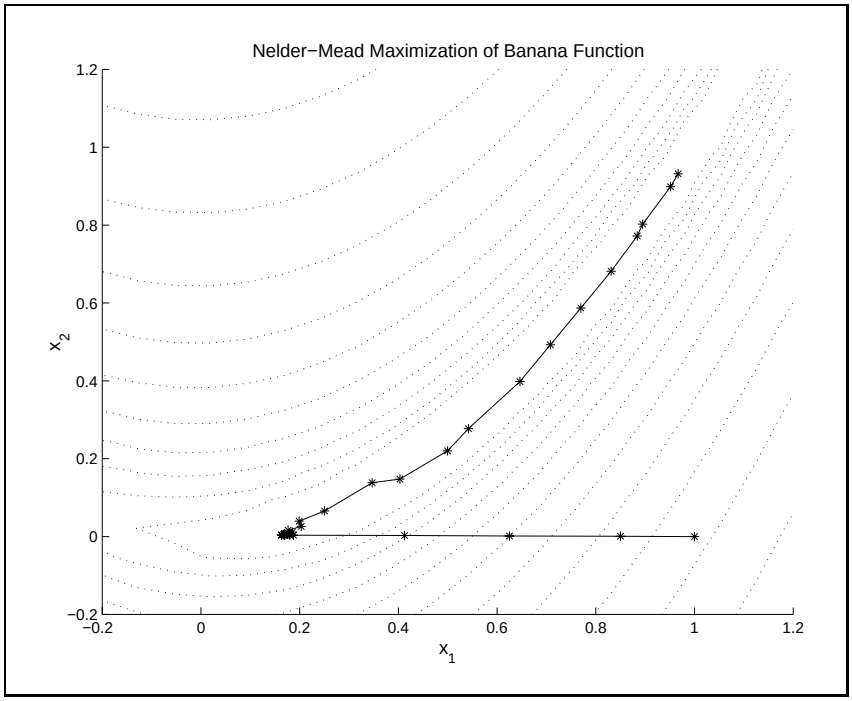


Figure 4.3

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 $x^{(k)}$:

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

Solving the first order condition

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

yields the iteration rule

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

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 $f''\left(x^{(k)}\right)$ 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 function (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

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

where $B^{(k)}$ 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 $f\left(x^{(k)} + sd^{(k)}\right)$. 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 inverse Hessian approximation $B^k$ is constructed and updated. The simplest quasi-Newton method sets $B^k = -I$, 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:

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

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:

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

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

$$B \leftarrow B + \frac{dd^\top}{d^\top u} - \frac{Buu^\top B}{u^\top B u},$$

where

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

and

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

The BFGS method uses the update scheme

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

where $w = d - Bu$.

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]

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, `searchmeth` (1-steepest ascent, 2-DFP, 3-BFGS).

```
k = size(x,1);
[fx0,g0] = f(x);
if all(abs(g0)<eps), return; end
for it=1:maxit
  d = -A*g0;                          % search direction
  [s,fx] = optstep(StepMeth,f,x,fx0,g0,d,maxstep,varargin{:});
  if fx<=fx0                          % Step search failure
    warning('Iterations stuck in qnewton'), return;
  end
  d = s*d;
  x = x+d;
  [fx,g] = f(x);
  % 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);
  elseif SearchMeth==2;                   % DFP update
```

[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 somewhat more complicated and requires routines to update Cholesky factors.

```
    v = A*u;
    A = A + d*d'./ud - v*v'./(u'*v);
  elseif SearchMeth==3;                    % BFGS update
    w = d-A*u; wd = w*d';
    A = A + ((wd + wd') - ((u'*w)*(d*d'))./ud)./ud;
  end
  %  Update iteration
  fx0 = fx; g0 = g;
end
```

Quasi-Newton methods are susceptible to certain problems. Notice in both 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 $\epsilon$ 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 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 $f(x) = -100\left(x_2 - x_1^2\right)^2 - \left(1 - x_1\right)^2$ assuming a starting value of $(1, 0)$. To maximize the function using the default BFGS 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.

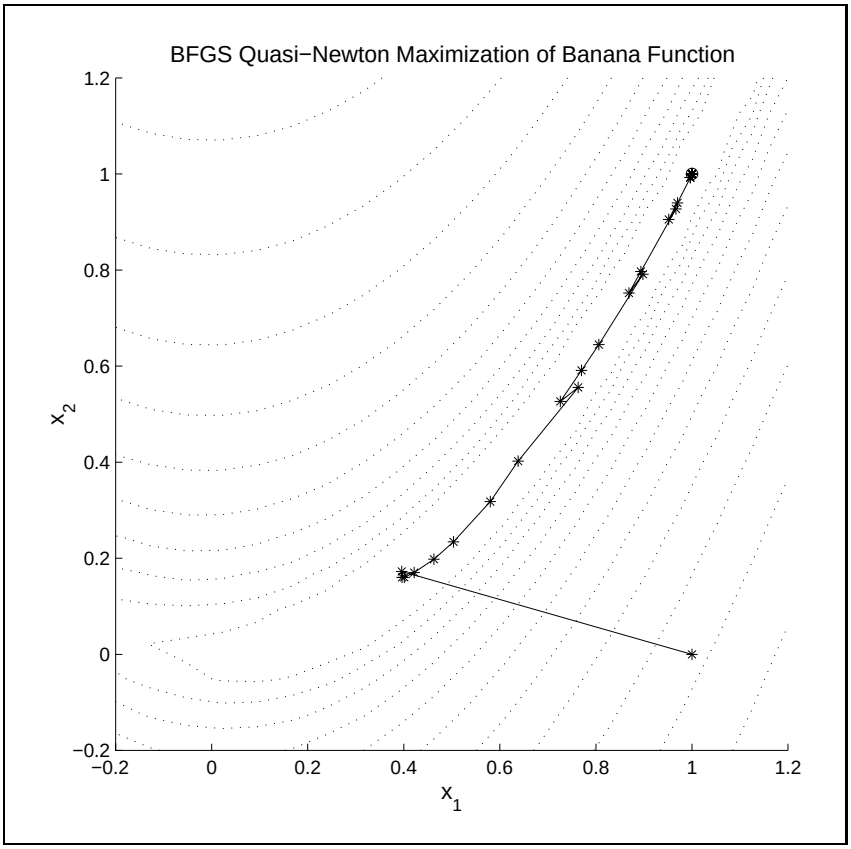


Figure 4.4

## 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 computationally 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

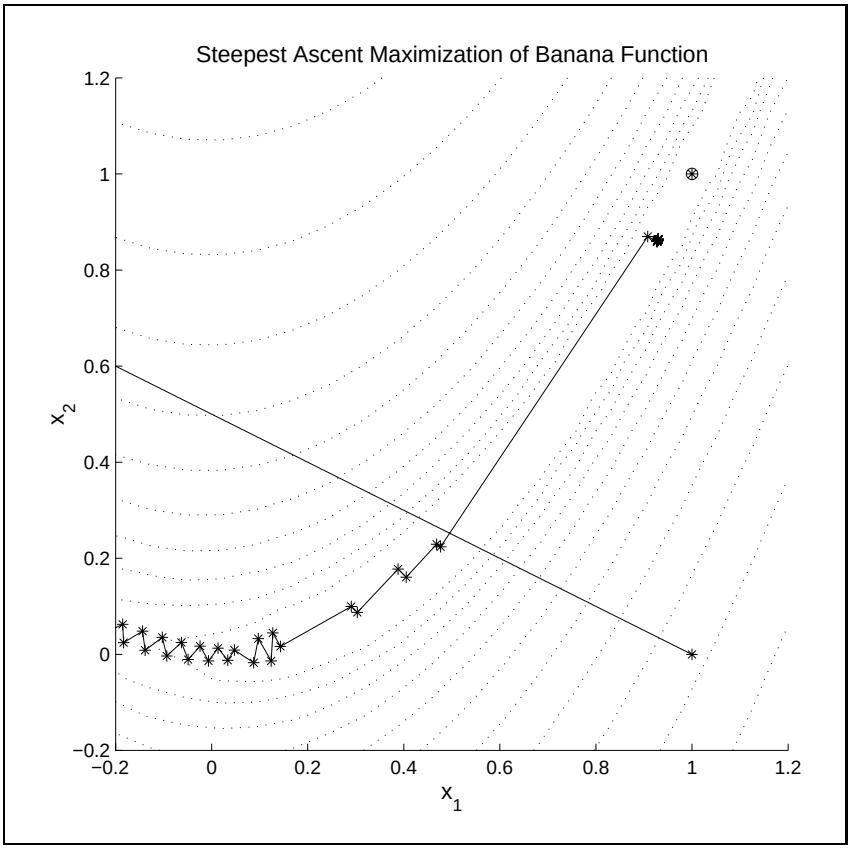


Figure 4.5

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, $\mu$ 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 $\mu_0$ and $\mu_1$ 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.

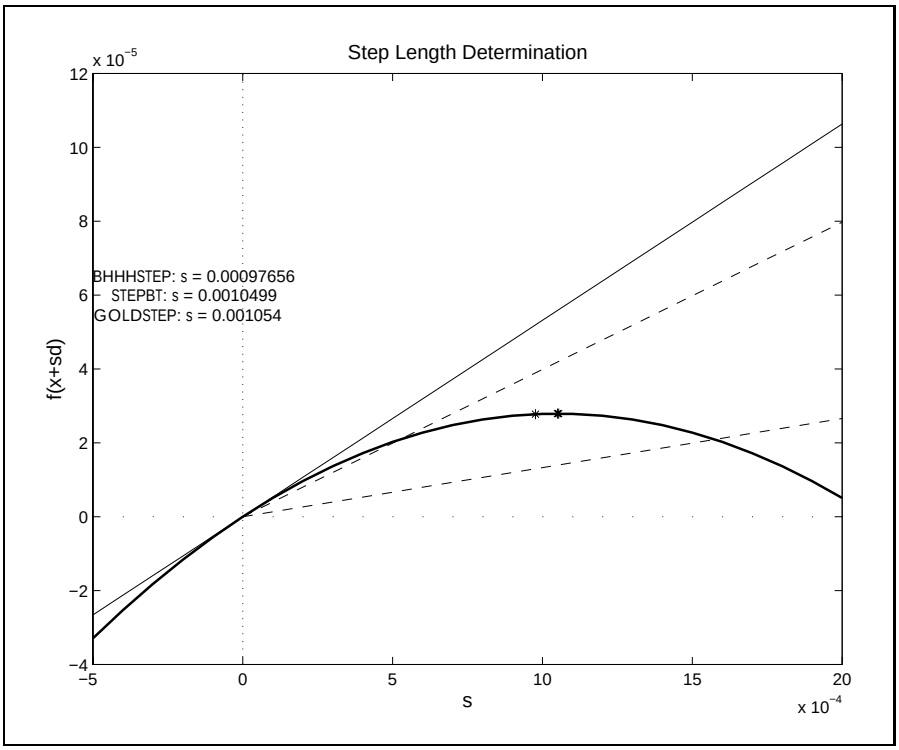


Figure 4.6

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.

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 $f(x)$, $f'(x)d$ and $f(x + d)$. If the computed step $s$ is acceptable, it is taken. Otherwise, the algorithm iterates until an acceptable step is found using a cubic approximation to the objective function in the Newton direction constructed from knowledge of $f(x)$, $f'(x)d$, $f(x + s^{(j-1)}d)$ and $f(x + s^{(j)}d)$. `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 terminates 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 cases, 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. Nonlinear 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 $\theta$ 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:

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

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, $f(\theta)$ composed of error terms each having expectation 0. Assuming that the usual central limit assumptions apply to the error term, the inverse of the approximate Hessian

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

can be used as a covariance estimator for $\theta$.

Maximum likelihood problems are specified by a choice of a distribution function for the data, $y$, that depends on a parameter vector, $\theta$. 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

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

where $\underline{1}_n$ is an $n$-vector of ones. This approach is known as the *modified method of scoring.*[4] As in the case of the nonlinear least squares, a covariance estimator for $\theta$ is immediately available using

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

## 4.6 Constrained Optimization

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

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

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 $\mathrm{CP}(f', a, b)$:[5]

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

Conversely, if $f$ is concave and differentiable on $[a, b]$ and $x^*$ solves the complementarity problem $\mathrm{CP}(f'(x), a, b)$, 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

[4] If the information matrix is known in closed form, it could be used rather than $s^\top s$ and the method would be known as the *method of scoring.*

[5] Complementarity problems are discussed in Section 3.7 on page 48.)

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.

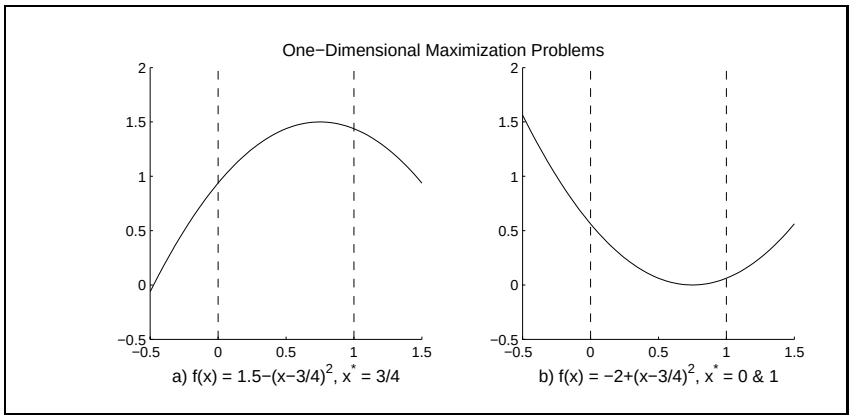


Figure 4.7

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 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.

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

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

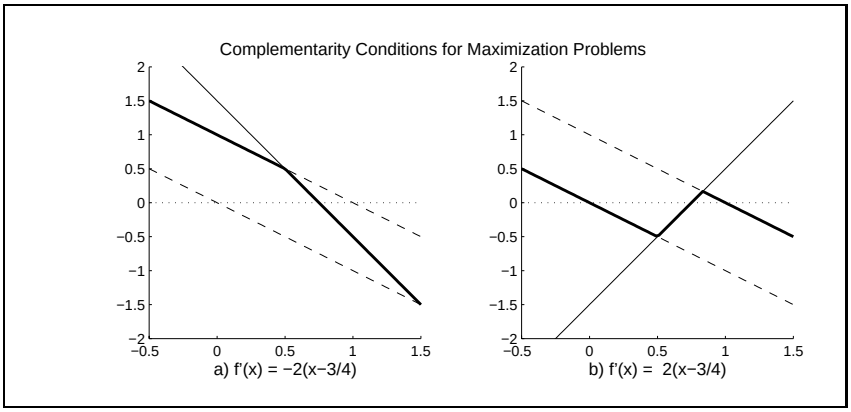


Figure 4.8

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

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

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 $(x, \lambda)$ solves the complementarity problem

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

where the values of $p$ and $q$ depend on the type of constraint:

| | $\leq$ | $=$ | $\geq$ |
|---|---|---|---|
| $p_i$ | $0$ | $-\infty$ | $-\infty$ |
| $q_i$ | $\infty$ | $\infty$ | $0$ |

A point $x$ is regular if the gradients of all constraint functions $R_i$ that satisfy $R_i(x) = r_i$ are linearly independent.[6] Conversely, if $f$ is concave, $R$ is convex and $(x, \lambda)$ satisfies the Karush-Kuhn-Tucker conditions, then $x$ solves the general constrained optimization problem.

[6]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.

In the Karush-Kuhn-Tucker conditions, the $\lambda_i$ 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, $\lambda_i$ 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 $\lambda_i^*$ 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**

4.1. 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 $X_i$ ($X_i$ is $1 \times k$ and $\beta$ is $k \times 1$).

(a) Show that the first order conditions for the maximum likelihood estimator of $\beta$ 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 $\beta$.

(c) Write a MATLAB function of the form `[beta,sigma]=example(y,X)` that computes the maximum likelihood estimator of $\beta$ and its asymptotic covariance matrix $\Sigma$. 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.

4.2. The two-parameter gamma probability distribution function has density:

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

(a) Derive the first order conditions associated with maximizing the log-likelihood associated with this distribution. Note that the first and second derivatives of the log of the $\Gamma$ function are the psi and trigamma functions. The MATLAB toolbox contains procedures to evaluate these special functions.

(b) Solve the first order condition for $\theta_2$ in terms of $\theta_1$. Use this to derive an optimality condition for $\theta_1$ alone.

(c) Write a MATLAB function that is passed a vector of observations (of positive numbers) and returns the maximum likelihood estimates of $\theta$ and their covariance matrix. Implement the function to use Newton's method without calling any general optimization or root-finding solvers.

Notice that the maximum likelihood estimator of $\theta$ 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 $\theta$). Your code should exploit this by only computing these sufficient statistics once.

(d) Plot $\theta_1$ as a function of $Y_1/Y_2$ over the range $[1.1, 3]$.

4.3. CIR Bond Pricing

The so-call Cox-Ingersoll-Ross (CIR) bond pricing model uses the function

$$Z(r, \tau; \kappa, \alpha, \sigma) = A(\tau)\exp(-B(\tau)r)$$

with

$$A(\tau) = \left(\frac{2\gamma e^{(\gamma+\kappa)\tau/2}}{(\gamma+\kappa)(e^{\gamma\tau}-1)+2\gamma}\right)^{2\kappa\alpha/\sigma^2}$$

and

$$B(\tau) = \frac{2(e^{\gamma\tau}-1)}{(\gamma+\kappa)(e^{\gamma\tau}-1)+2\gamma},$$

where $\gamma = \sqrt{\kappa^2 + 2\sigma^2}$. Here $r$ is the current instantaneous rate of interest, $\tau$ is the time to maturity of the bond, and $\kappa$, $\alpha$ and $\sigma$ are model parameters. The percent rate of return on a bond is given by

$$r(\tau) = -100\ln(Z(r, \tau))/\tau.$$

In the following table, actual rates of return[7] on Treasury bonds for 9 values of $\tau$ are given for 5 consecutive Wednesdays in early 1999.

| Date | .25 | .5 | 1 | 2 | 3 | 5 | 7 | 10 | 30 |
|---|---|---|---|---|---|---|---|---|---|
| 1999/01/07 | 4.44 | 4.49 | 4.51 | 4.63 | 4.63 | 4.62 | 4.82 | 4.77 | 5.23 |
| 1999/01/13 | 4.45 | 4.48 | 4.49 | 4.61 | 4.61 | 4.60 | 4.84 | 4.74 | 5.16 |
| 1999/01/20 | 4.37 | 4.49 | 4.53 | 4.66 | 4.66 | 4.65 | 4.86 | 4.76 | 5.18 |
| 1999/01/27 | 4.47 | 4.47 | 4.51 | 4.57 | 4.57 | 4.57 | 4.74 | 4.68 | 5.14 |
| 1999/02/03 | 4.48 | 4.55 | 4.59 | 4.72 | 4.73 | 4.74 | 4.91 | 4.83 | 5.25 |

[7]Actually, the data is constructed by a smoothing and fitting process and thus these returns do not necessarily represent the market prices of actual bonds; for the purposes of the exercise, however, this fact can be ignored.

a) For each date, find the values of $r$, $\kappa$, $\alpha$ and $\sigma$ that minimize the squared differences between the model and the actual rates of return. This is one way that model parameters can be "calibrated" to the data and ensures that model parameters yield a term structure that is close to the observed term structure.

b) In this model the values of the parameters are fixed, but the value of $r$ varies over time. In fact, part (a) showed that the three parameters values vary from week to week. As an alternative, find the values of the parameters and the 5 values of $r$ that minimize the squared deviations between the model and actual values. Compare these to the parameter values obtained by calibrating to each date separately.

4.4. Option-based Risk-neutral Probabilities

An important theorem in finance theory demonstrates that the value of a European put option is equal to the expected return on the option, with the expectation taken with respect to the so-called risk-neutral probability measure[8]

$$V(k) = \int_0^\infty (k-p)^+ f(p)dp = \int_0^k (k-p)f(p)dp$$

where $f(p)$ is the probability distribution of the price of underlying asset at the option's maturity, $k$ is the option's strike price and $(x)^+ = \max(0, x)$.

This relationship has been used to compute estimates of $f(p)$ based on observed asset prices. There are two approaches that have been taken. The first is to choose a parametric form for $f$ and find the parameters that best fit the observed option price. To illustrate, define the discrepancy between observed and model values as

$$e(k) = V_k - \int_0^k (k-p)f(p;\theta)dp$$

and then fit $\theta$ by, e.g., minimizing the sum of squared errors:

$$\min_\theta \sum_j e(k_j)^2.$$

The other approach is to discretize the price, $p_i$, and its probability distribution, $f_i$. Values of the $f_i$ can be computed that correctly reproduce observed option

[8]This is strictly true only if the interest rate is 0 or, equivalently, if the option values are interest rate adjusted appropriately. Also, the price of the underlying asset should not be correlated with the interest rate.

value and that satisfy some auxiliary condition. That condition could be a smoothness condition, such as minimizing the sum of the $f_{i+1} - 2f_i + f_{i-1}$; if the $p_i$ are evenly spaced this is proportional to an approximation to the second derivative of $f(p)$.

An alternative is to compute the maximum entropy values of the $f_i$:

$$\max_{\{f_i\}} \sum_i f_i \ln(f_i),$$

subject to the constraints that the $f_i$ are non-negative and sum to 1. It is easy to show that the $f_i$ satisfy

$$f_i = \frac{\exp(\sum_j \lambda_j (k_j - p_i)^+)}{\sum_i \exp(\sum_j \lambda_j (k_j - p_i)^+)},$$

where $\lambda_j$ is the Lagrange multiplier on the constraint that the $j$th option is correctly priced. The problem is thus converted to the root finding problem of solving for the Lagrange multipliers:

$$V_j - \sum_i f_i (k_j - p_i)^+ = 0,$$

where the $f_i$ are given above.

Write a MATLAB program that takes as input a vector of price nodes, $p$, and associated vectors of strike prices, $k$, and observed put option values, $v$, and returns a vector of maximum entropy probabilities, $f$, associated with $p$:

```
f=RiskNeutral(p,k,v)
```

The function can pass an auxiliary function to a root finding algorithm such as `Newton` or `Broyden`.

The procedure just described has the peculiar property that (if put options alone are used), the upper tail probabilities are all equal above the highest value of the $k_j$. To correct for this, one can add in the constraint that the expected price at the option's expiration date is the current value of the asset, as would be true in a 0 interest rate situation. Thus modify the original program to accept the current value of the price of the underlying asset:

```
f=RiskNeutral(p,k,v,p0)
```

To test your program, use the script file RiskNeutD.m.

4.5. Consider the *Quadratic Programming* problem

$$\begin{array}{ll} \max\limits_{x} & \frac{1}{2}x^\top Dx + c^\top x \\ s.t. & Ax \leq b \\ & x \geq 0 \end{array}$$

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?

4.6. 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.

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

$$\begin{aligned} \min \quad & \sum_{i=1}^{n}\sum_{j=1}^{m} x_{ij}x_{ij} \\ s.t. \quad & \sum_{i=1}^{n} x_{ij} \geq d_j \quad j = 1,\ldots,m \\ & \sum_{j=1}^{m} x_{ij} \leq s_i \quad i = 1,\ldots,n \\ & x_{ij} \geq 0 \qquad i = 1,\ldots,n, j = 1,\ldots,m \end{aligned}$$

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

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

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

Supply is given by:

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

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.

4.9. Portfolio Choice

Suppose that the returns on a set of $n$ assets has mean $\mu$ $(n \times 1)$ and variance $\Sigma$ $(n \times n)$. A portfolio of assets can be characterized by a set of share weights, $\omega$, an $n \times 1$ vector of non-negative values summing to 1. The mean return on portfolio is $\mu^\top\omega$ and its variance is $\omega^\top\Sigma\omega$.

A portfolio is said to be on the mean-variance efficient frontier if its variance is as small as possible for a given mean return.

Write a MATLAB program that calculates and plots a mean-variance efficient frontier. Write it so it returns two vectors that provide points on the frontier:

```
[mustar,Sigmastar]=MV(mu,Sigma,n)
```

Here $n$ represents the desired number of points.

Run the program MVDemo.m to test your program.

Hint: Determine the mean return from the minimium variance portfolio and determine the maximum mean return portfolio. These provide lower and upper bounds for `mustar`. Then solve the optimization problem for the remaining $n-2$ values of `mustar`.

4.10. Consider the nonlinear programming problem

$$\begin{array}{ll} \max\limits_{x_1,x_2} & x_2^2 - 2x_1 - x_1^2 \\ s.t. & x_1^2 + x_2^2 \leq 1 \\ & x_1 \geq 0, x_2 \geq 0. \end{array}$$

(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.

4.11. 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.

4.12. 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.

4.13. 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

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

Letting $\lambda$ 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 $\lambda^*$ and interpret its meaning.

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

4.14. 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.

4.15. 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?

4.16. 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 $\lambda_1$ and $\lambda_2$ associated with the structural constraints, and interpret $\lambda_1^*$ and $\lambda_2^*$.

(c) If the second constraint were changed to $x_1 + x_2 \leq 5$, what would be the effect on the optimal values of $x_1$, $x_2$, $\lambda_1$, and $\lambda_2$?

## Bibliographic Notes

A number of very useful references exist on computational aspects of optimization. Perhaps the most generally useful for practitioners are Gill et al. and Fletcher. Ferris and Sinapiromsaran discusses solving non-linear optimization problems by formulating them as CPs.

# 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 $f(\tilde{X})$ when $I$ repesents the whole support of $\tilde{X}$.

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 are especially useful if the integration domain is of high dimension or irregularly shaped.

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

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

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

$$\int_a^b f(x)\ dx \approx \sum_{i=1}^{n} w_i\ f(x_i)$$

where $w_1 = w_n = h/2$ 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 $O(h^2)$, that is, the error shrinks quadratically with the size of the sampling interval.

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

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

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

$$\int_a^b f(x)\, dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

where $w_1 = w_n = h/3$ and, otherwise, $w_i = 4h/3$ if $i$ is odd and $w_i = 2h/3$ if $i$ is even.

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 $O(h^4)$, 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 nodes 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 $n = n_1 n_2$ grid points of the form $\{(x_{1i}, x_{2j})|i = 1, 2, \ldots, n_1; j = 1, 2, \ldots, n_2\}$ with associated weights $\{w_{ij} = w_{1i} w_{2j}|i = 1, 2, \ldots, n_1; j = 1, 2, \ldots, n_2\}$. 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, $w(x) \equiv 1$, 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 $n$.

Selected Newton-Cotes and Gaussian quadrature methods are compared in Table 5.1. The table illustrates that 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, 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.

**Table 5.1: Errors for Selected Quadrature Methods**

| Function | Degree ($n$) | 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 |

When the weighting function $w(x)$ 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:

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

For example, the three-point approximation to the standard univariate normal distribution $\tilde{Z}$ is characterized by the condition that moments 0 through 5 match those of the standard normal: $E\tilde{Z}^0 = 1$, $E\tilde{Z}^1 = 0$, $E\tilde{Z}^2 = 1$, $E\tilde{Z}^3 = 0$, $E\tilde{Z}^4 = 3$, 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 related to Gauss-Hermite quadrature.[1]

As was the case with Newton-Cotes quadrature, tensor product principles may be applied to univariate Gaussian quadrature rules to develop quadrature rules for multivariate integration. Suppose, for example, that $\tilde{X}$ is a $d$-dimensional normal random variable with mean vector $\mu$ and variance-covariance matrix $\Sigma$. Then $\tilde{X}$ is distributed as $\mu + \tilde{Z}R$ where $R$ is the Cholesky square root of $\Sigma$ (e.g., $\Sigma = R^\top R$) 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 an $n^d$ degree

[1]Gauss-Hermite quadrature applies to the weighting function $w(x) = \exp(-x^2)$, as opposed the weighting function for the standard normal density $w(x) = \exp(-x^2/2)/\sqrt{2\pi}$.

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 $\mu$ and $\sigma^2$ if, and only if, it is distributed as $\exp(\tilde{X})$ were $\tilde{X}$ is normally distributed with mean $\mu$ and variance $\sigma^2$. It follows that if $\{x_i, w_i\}$ are nodes and weights for a Normal$(\mu, \sigma^2)$ distribution, then $\{y_i, w_i\}$, where $y_i = \exp(x_i)$, provides a reasonable discrete approximant for a Lognormal$(\mu, \sigma^2)$ 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 expectation of $f(\tilde{X})$, one draws a random sample $x_1, x_2, \ldots, x_n$ from the distribution of $\tilde{X}$ and sets

$$E\left[f(\tilde{X})\right] \approx \frac{1}{n} \sum_{i=1}^{n} f(x_i).$$

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(m,n)` or `x=randn(m,n)` generates a random sample of $mn$ realizations and stores it in an $m \times n$ matrix.

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

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

has the desired distribution $F$. 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 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 of nodes and weights, `a` is the vector of left endpoints, `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 these sequences and a pseudo-random sequence are illustrated in Figure 5.1. Each of the plots shows $4,000$ 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]

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

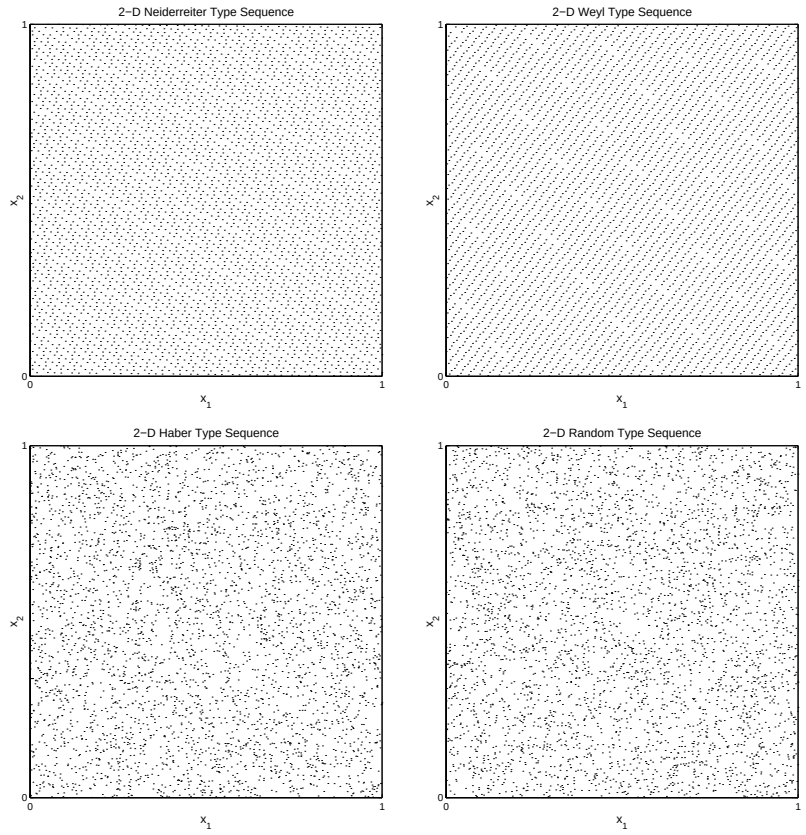


Figure 5.1: Alternative Equidistributed Sequences

**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 |

## 5.5 An Integration Toolbox

The MATLAB toolbox accompanying the textbook includes four functions for computing numerical integrals for general functions. Each takes three inputs, `n`, `a`, and `b` and generates appropriate nodes and weights. The functions `qnwtrap` and `qnwsimp` imple-

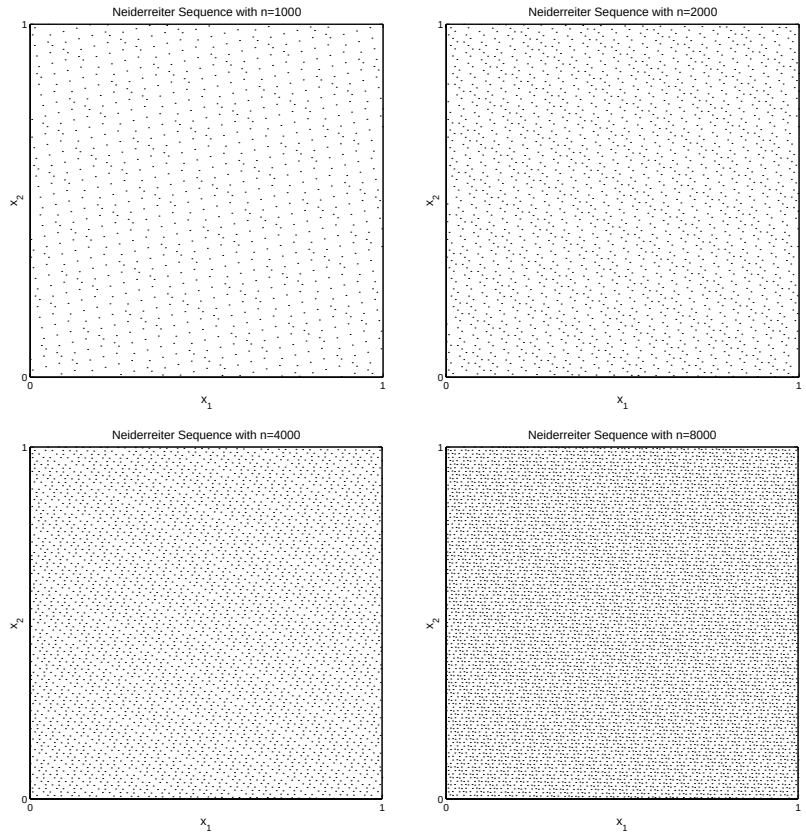


Figure 5.2: Fill in of the Neiderreiter Sequence

ment the Newton-Cotes trapezoid and Simpson's rule methods, `qnwlege` implements Gauss-Legendre quadrature and `qnwequi` generates nodes and weights associated with either equidistributed or pseudo-random sequences. The calling syntax is the same for each and is illustrated with below with `qnwtrap`.

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

The inputs are is the number nodes and weights, `n`, the left endpoint, `a` and the right endpoint, `b`. The outputs are the nodes, `x`, and the weights, `w`. 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);
```

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; 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.

All of the quadrature functions will use tensor products to generate nodes and weights for integration over an arbitrary bounded interval $[a, b]$ in higher dimensional spaces. For a $d$-variable function, with $n_i$ nodal points for the $i$th variable, $w$ is $n \times 1$ and $x$ is $n \times d$, where $n = \prod_{i=1}^{d} n_i$. For example, suppose one wished to compute the integral of $\exp(x_1 + x_2)$ over the rectangle $[1, 2] \times [0, 5]$ in $\Re^2$. One could invoke `qnwtrap` to construct a grid of, say, 2601 quadrature nodes produced 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. Using `qnwlege` with 5 nodes in the $x_1$ direction and 4 nodes in the $x_2$ direction:

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

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.

In addition to the general integration routines, the MATLAB toolbox accompanying the textbook also includes several functions for computing nodes and weights associated with common distribution functions. `qnwnorm` generates the 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` also generates 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 $[2 \quad -1; -1 \quad 4]$. One could invoke `qnwnorm` to construct a grid of 100 Gaussian quadrature nodes as the cross-product of 10 nodes in the $x_1$ direction and 10 nodes 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]);
expectation = w'*exp(x(:,1)+x(:,2));
```

This computation would yield an approximate answer of 8103.083, which is accurate to 7 significant digits (the exact value is $e^9$).

Other quadrature functions included in the MATLAB toolbox accompanying the textbook generate quadrature nodes and weights for computing the expectations of functions of lognormal, beta and gamma random variates. For univariate lognormal distributions, the calling sequence takes the form

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

where `mu` and `var` are the mean and variance of the log of $x$. For the beta distribution, the calling syntax is

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

where `a` and `b` are the shape parameters of the beta distribution. For the gamma distribution, the calling syntax is

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

where `a` is the shape parameters of the (one dimensional) gamma distribution. For both the beta and gamma distributions the parameters may be passed as vectors, yielding nodes and weights for multivariate *independent* random variables.

In addition to the quadrature routines provided with this book, MATLAB offers two Newton-Cotes quadrature routines, `quad` and `quad8`, both of which employ an adaptive Simpson's rule.

## 5.6 Numerical Differentiation

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

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

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. We will return to the question of how small $h$ should be, but first we address the issue of how large an error is produced using this finite difference approach.

An error bound for the approximation can be be obtained using a Taylor expansion. 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 the approximation to the derivative $f'(x)$ has an $O(h)$ error.

The simple $O(h)$ approximation is a two-point approximation, meaning that only two function values are used. In order to obtain more accurate approximations, consider evaluating the function at three points, $x$, $x+h$ and $x+\lambda h$ and approximating the derivative with a weighted sum of these values:

$$f'(x) \approx af(x) + bf(x+h) + cf(x+\lambda h).$$

To determine both the appropriate values of $a$, $b$, and $c$ and to determine the size of the approximation error, expand the Taylor series for $f(x+h)$ and $f(x+\lambda h)$ around $x$, obtaining

$$\begin{aligned} af(x) + bf(x+h) + cf(x+\lambda h) = \\ (a+b+c)f(x) + h(b+c\lambda)f'(x) + \tfrac{h^2}{2}(b+c\lambda^2)f''(x) \\ + \tfrac{h^3}{6}\Big(bf^{(3)}(z_1) + c\lambda^3 f^{(3)}(z_2)\Big). \end{aligned}$$

(for some $z_1 \in [x, x+h]$ and $z_2 \in [x, x+\lambda h]$). The constraints $a+b+c = 0$, $b + c\lambda = 1/h$ and $b + c\lambda^2 = 0$ uniquely determine $a$, $b$ and $c$:

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \frac{1}{h\lambda(1-\lambda)} \begin{bmatrix} \lambda^2 - 1 \\ -\lambda^2 \\ 1 \end{bmatrix}$$

leading to

$$af(x) + bf(x+h) + cf(x+\lambda h) = f'(x) + O(h^2).$$

Thus, by using 3 points, we can ensure that the approximation converges at a quadratic rate in $h$.

Some special cases of importance arise when the evaluations points are evenly spaced. When $\lambda = -1$, $x$ lies halfway between the other points and we obtain the centered finite difference approximation

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

which is second order accurate even though only two approximation points are used. If $\lambda = 2$, we obtain a formula that is useful when a derivative is needed at a boundary of a domain. In this case

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

(use $h > 0$ for a lower bound and $h < 0$ for an upper bound).

To obtain formulii for second derivatives we can use the same approach but in order to obtain second order accuracy, we will (in general) require a weighted sum composed of 4 points

$$f'(x) \approx af(x) + bf(x+h) + cf(x+\lambda h) + df(x+\psi h).$$

We also expand the Taylor series to the fourth order, obtaining

$$\begin{aligned} &af(x) + bf(x+h) + cf(x+\lambda h) + df(x+\psi h) = \\ &\quad (a+b+c+d)f(x) + h(b + c\lambda + d\psi)f'(x) + \tfrac{h^2}{2}(b + c\lambda^2 + d\psi^2)f''(x) \\ &\quad + \tfrac{h^3}{6}(b + c\lambda^3 + d\psi^3)f'''(x) + \tfrac{h^4}{24}\Big(bf^{(4)}(z_1) + c\lambda^4 f^{(4)}(z_2) + d\psi^4 f^{(4)}(z_3)\Big). \end{aligned}$$

The constraints $a + b + c + d = 0$, $b + c\lambda + d\psi = 0$, $b + c\lambda^2 + d\psi^2 = 2/h^2$ and $b + c\lambda^3 + d\psi^3 = 0$ uniquely determine $a$, $b$, $c$ and $d$:

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \frac{2}{h^2} \begin{bmatrix} \dfrac{1+\lambda+\psi}{\lambda\psi} \\ \dfrac{\lambda^2 - \psi^2}{(\lambda-1)(\psi-1)(\psi-\lambda)} \\ \dfrac{\psi^2 - 1}{\lambda(\lambda-1)(\psi-1)(\psi-\lambda)} \\ \dfrac{1-\lambda^2}{\psi(\lambda-1)(\psi-1)(\psi-\lambda)} \end{bmatrix}$$

with

$$af(x) + bf(x+h) + cf(x+\lambda h) + df(x+\psi h) = f''(x) + O(h^2).$$

Thus, by using 4 points, we can ensure that the approximation converges at a quadratic rate in $h$.

Some special cases of importance arise when the evaluations points are evenly spaced. When $x$ lies halfway between $x+h$ and one of the other two points (i.e., when either $\lambda = -1$ or $\psi = -1$), we obtain the centered finite difference approximation

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

which is second order accurate even though only three approximation points are used. If $\lambda = 2$ and $\psi = 3$ we obtain a formula that is useful when a derivative is needed at a boundary of the domain. In this case

$$f''(x) = \frac{1}{h^2}[2f(x) - 5f(x+h) + 4f(x+2h) - f(x+3h)] + O(h^2).$$

An important use of second derivatives is in computing Hessian matrices. Given some function $f : \Re^n \to \Re$, the Hessian is the $n \times n$ matrix of second partial derivatives, the $ij$th element of which is $\frac{\partial^2 f(x)}{\partial x_i \partial x_j}$. We consider only centered, evenly spaced approximations, which can be obtained as a weighted sum of the function values evaluated at the point $x$ and 8 points surrounding it obtained by adding or subtracting $h_i u_i$ and/or $h_j u_j$, where the $h$ terms are scalar step increments and the $u$ terms are $n$-vectors of zeros but with the $i$th element equal to 1 (the $i$th column of $\mathrm{I}_n$).

To facilitate notation, let subscripts indicate a partial derivative of $f$ evaluated at $x$, e.g., $f_i = \frac{\partial f(x)}{\partial x_i}$, $f_{iij} = \frac{\partial^3 f(x)}{\partial x_i^2 \partial x_j}$, etc. and let superscripts on $f$ denote the function evaluated at one of the 9 points of interest, so $f^{++} = f(x + h_i u_i + h_j u_j)$, $f^{00} = f(x)$, $f^{0-} = f(x - h_j u_j)$, etc. (see Figure 5.3).

With this notation, we can write Taylor expansions up to the third order for each of the $f^{ij}$. For example

$$f^{+0} = f^{00} + h_i f_i + \frac{h_i^2}{2} f_{ii} + \frac{h_i^3}{6} f_{iii} + O(h^4),$$

and

$$\begin{aligned} f^{++} \;=\; & f^{00} + h_i f_i + h_j f_j + \tfrac{h_i^2}{2} f_{ii} + h_i h_j f_{ij} \tfrac{h_j^2}{2} f_{jj} \\ & + \tfrac{h_i^3}{6} f_{iii} + \tfrac{h_i^2 h_j}{2} f_{iij} + \tfrac{h_i h_j^2}{2} f_{ijj} + \tfrac{h_j^3}{6} f_{jjj} + O(h^4). \end{aligned}$$

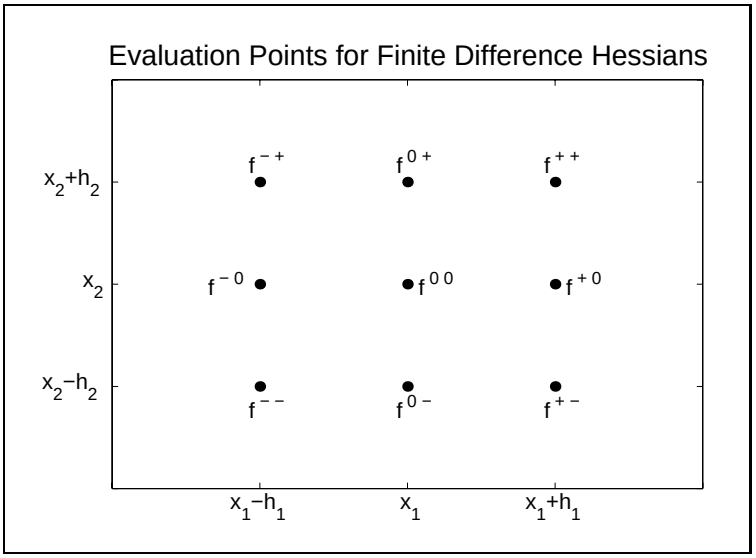


Figure 5.3

With simple but tedious computations, it can be shown that the only $O(h^2)$ approximations to $f_{ii}$ composed of these 9 points are convex combinations of the usual centered approximation

$$f_{ii} \approx \frac{1}{h_i^2}(f^{+0} - 2f^{00} + f^{-0})$$

and an alternative

$$f_{ii} \approx \frac{1}{2h_i^2}(f^{++} - 2f^{0+} + f^{-+} + f^{-+} - 2f^{-0} + f^{--}).$$

More importantly, for computing cross partials, the only $O(h^2)$ approximations to $f_{ij}$ are convex combinations of

$$f_{ij} \approx \frac{1}{2h_ih_j}(f^{0+} + f^{-0} + f^{0-} + f^{+0} - f^{+-} - f^{-+} - 2f^{00})$$

or

$$f_{ij} \approx \frac{1}{2h_ih_j}(2f^{00} + f^{++} + f^{--} - f^{0+} - f^{-0} - f^{0-} - f^{+0}).$$

The obvious combination of taking the mean of the two results in

$$f_{ij} \approx \frac{1}{4h_ih_j}(f^{++} + f^{--} - f^{-+} - f^{+-}).$$

This requires less computation than the other two forms if only a single cross partial is evaluated. Using either of the other two schemes, however, along with the usual centered approximation for the diagonal terms of the Hessian enables one to compute the entire Hessian with second order accuracy in $1 + n + n^2$ function evaluations.

There are typically two situations in which numerical approximations of derivatives are needed. The first arises when one can compute the function at any value of $x$ but it is difficult to derive an closed form expression for the derivatives. In this case one is free to choose the evaluation points ($x$, $x+h$, $x+\lambda h$, etc.). The other situation is one in which the value of $f$ is known only at a fixed set of points $x_1$, $x_2$, etc. This situation arises frequently in interpolation and functional equation problems, which we consider in the next chapter (see especially Section 6.4, page 143).

When a function can be evaluated at any point, the choice of evaluation points must be considered. 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 5.4a, 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 5.4b shows an analogous plot for two-sided approximations. It is evident that the error is minimized at a much higher value of $h$, at approximately $\sqrt[3]{\epsilon}$. A good rule of thumb is to set

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

when using two-sided approximations.

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

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

Even if the rounding error $e$ is on the order of machine accuracy, $\epsilon$, and $h$ on the order of $\sqrt{\epsilon}$, we have introduced an error on the order of $\sqrt{\epsilon}$ into the calculation. It

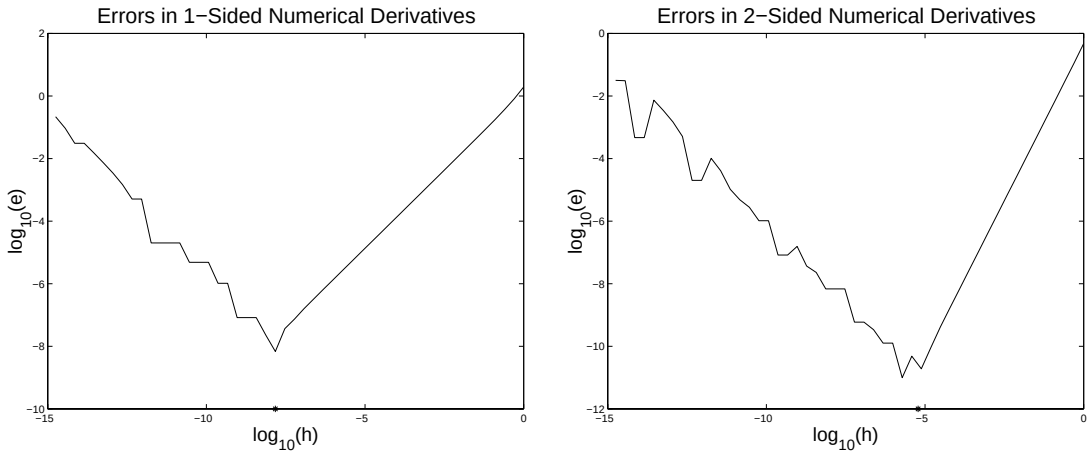


Figure 5.4

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); xh=x+h; h=xh-x;
```

for one-sided and

```
h=eps.^(1/3)*max(x,1); xh1=x+h; xh0=x-h; hh=xh1-xh0;
```

for two-sided approximations (`hh` represents $2h$).

We provide below a function that computes two-sided finite difference approximations for the Jacobian of an arbitrary function. For a real-valued function, $f : \Re^n \mapsto \Re^m$, the output is an $m \times n$ matrix:

```
function fjac = fdjac(f,x);
  h = eps^(1/3)*max(abs(x),1);
  xh1=x+h; xh0=x-h; h=xh1-xh0;
  for j=1:length(x);
    xx = x;
    xx(j) = xh1(j); f1=feval(f,xx);
    xx(j) = xh0(j); f0=feval(f,xx);
    fjac(:,j) = (f1-f0)/h(j);
  end
```

For second derivatives, the choice of $h$ encounters the same difficulties as with first derivatives and similar reasoning leads to the rule of thumb that

$$h = \max(x, 1)\sqrt[4]{\epsilon}$$

A procedure for computing finite difference Hessians, `fdhess` is provided in the CompEcon toolbox. It is analogous to `fdjac`, with calling syntax

```
fhess = fdhess(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. For ODEs, the solution of an IVP is known at some point and the solution near this point 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 until the next chapter (page 164).

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

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

Here, $x$ is a function of a scalar $t$ (often referring to time in economic applications) and $f : [0,T] \times \Re^d \mapsto \Re^d$ 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

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

By defining $z$ to be the first derivative of $x$, so that $z' = x''$, the differential equation may be written in first order form

$$y' = z$$

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

Initial value problems can be solved using a recursive 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, $n$ (or, equivalently, the stopping point $T$).

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 $t_i$

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

with the procedure beginning at the prescribed $x_0 = x(0)$. This method is fine for rough approximations, especially if the time step is small enough. Higher order approximations can yield better results, however.

Among the numerous refinements on the Euler method, the most commonly used are the Runge-Kutta methods. Runge-Kutta methods are 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 $t$.

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

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

$$f(t + \lambda h, x + \lambda h f) = f + \lambda h (f_t + f_x f) + O(h^2).$$

Substituting this into the previous expression yields

$$x(t + h) = x + h\left[\left(1 - \frac{1}{2\lambda}\right) f(t, x) + \frac{1}{2\lambda} f\Big(t + \lambda h, x + \lambda h f\Big)\right] + O(h^3). \quad (5.1)$$

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

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

and

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

It can be shown that an optimal choice, in the sense of minimizing the absolute value of the $h^3$ term in the truncation error, is to set $\lambda = 2/3$:

$$x(t+h) \approx x + \frac{h}{4}\left[f(t,x) + 3f\left(t + \frac{2h}{3}, x + \frac{2h}{3}f\right)\right]$$

(we leave this as an exercise)

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

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

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

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

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

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

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

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

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

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

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

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 implements the classical fourth order Runge-Kutta approach to compute an approximate solution $x(T)$ to $x' = f(t, x)$, s.t. $x(T(1)) = x0$, where $T$ is a vector of values. The calling syntax is

```
x=rk4(f,T,x0,[],additional parameters)
```

The inputs are the name of a problem file that returns the function $f$, the vector of time values $T$ and the initial conditions, $x_0$. The fourth input is an empty matrix to make the calling syntax for `rk4` compatible with MATLAB's ODE solvers. Unlike the suite of ODE solvers provided by MATLAB, `RK4` is designed to compute solutions for multiple initial values. If $x0$ is $d \times k$ and there are $n$ time values in $T$, RK4 will return an $n \times d \times k$ array. Avoiding a loop over multiple starting points results in much faster execution when a large set of trajectories are computed. To take advantage of this feature, however, the function 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 the example 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 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.[3]

**Example: Commercial Fishery**

As an example of an initial value problem, consider the following model of a com-

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

mercial fishery:

| | |
|---|---|
| $p = \alpha - \beta K y$ | 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 |

where $p$ is the price of fish, $K$ is the size of the industry, $y$ is the catch rate of the representative firm, $\pi$ is the profit of the representative firm and $S$ is the fish population ($\alpha$, $\beta$, $c$, $f$, $a$, $b$ and $\delta$ are parameters).

The behavior of this model can be analyzed by first determining 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 industry is competitive in the sense that catch rates are chosen by setting marginal cost equal to price:

$$p = cy/S,$$

a relationship that can be interpreted 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.

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. The capital stock adjustment process is myopic,

as it depends only on current profitability and not on expected future profitability. The result is a 2 dimensional IVP:

$$\begin{aligned} S' &= (a - bS)S - \frac{\alpha SK}{c + \beta SK} \\ K' &= \delta \left( \frac{c\alpha^2 S}{2(c + \beta SK)^2} - f \right) \end{aligned}$$

which can be solved for any initial fish stock ($S$) and industry size ($K$).

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 is exhibited in Figure 5.5 for parameter values $\beta = 2.75$, $f = 0.06$, $\delta = 10$ and other parameters normalized to equal 1. The so-called zero-isoclines (the points in the state space for which one of the variables' time rate of change is zero) are shown as dashed lines. In the phase diagram in Figure 5.5, the dashed lines represent the zero-isoclines and the solid lines the trajectories.

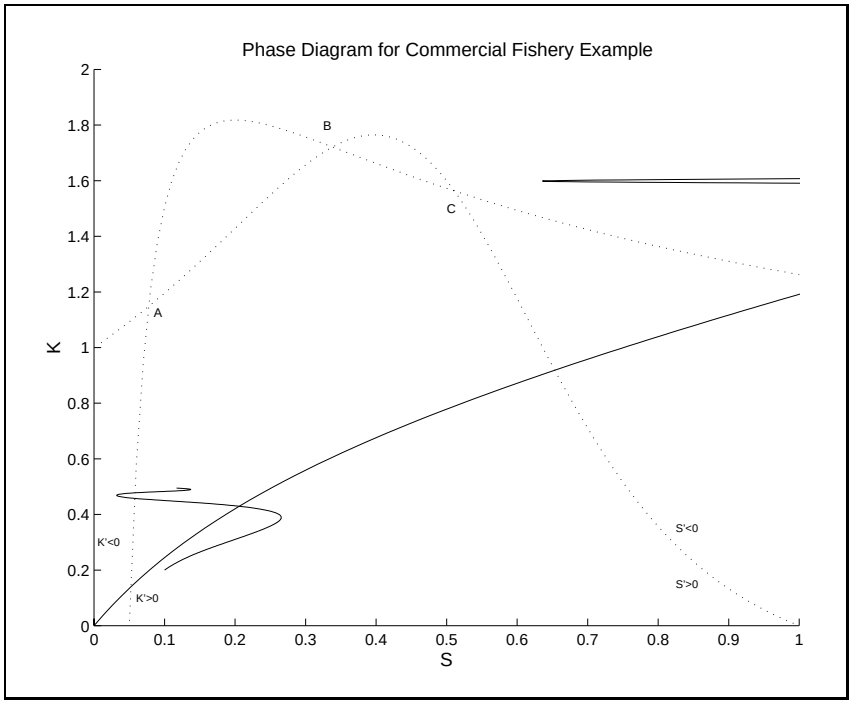


Figure 5.5

There are 3 long-run equilibria in this system; these are the points where the zero-isoclines cross. Two of the equilibria are locally stable (points A and C) and one

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. Also note that point A exhibits cyclic convergence.

**Exercises**

5.1. 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.

5.2. For $z > 0$, the cumulative probability function for a standard normal random variable is given by

$$F(z) = 0.5 + \frac{1}{\sqrt{2\pi}} \int_0^z \exp\left(-\frac{x^2}{2}\right)\ dx.$$

(a) Write a short MATLAB program that computes 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?

5.3. Write a MATLAB program that solves numerically the following expression for $\alpha$:

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

and demonstrate that the solution (to 4 significant digits) is $\alpha = 0.5061$.

5.4. 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.

5.5. 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 $E[p]$ and the ex-ante variance of price $V[p]$ using a $6^{th}$ degree Gaussian discretization for the demand shocks.

(b) Compute the expected price $E[p]$ and the ex-ante variance of price $V[p]$ 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.

5.6. Consider the commodity market model of Chapter 1 (page 2), 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.

5.7. Consider a market for an agricultural commodity in which farmers receive a government payment whenever the market price $p$ drops below an announced target price $\bar{p}$: $\max(\bar{p}-p, 0)$. 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:

$$a = 1 + (E[f])^{0.5}.$$

Production $q$ is acreage planted $a$ times a random yield $\tilde{y}$, unknown at planting time:

$$q = a\tilde{y};$$

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 $E[f]$;

(c) the ex-ante variance of producer price $V[f]$;

(d) the ex-ante expected producer revenue $E[fq]$; and

(e) the ex-ante variance of producer revenue $V[fq]$.

5.8. Suppose acreage planted at the beginning of the growing season is given by $a = \alpha(E[p], V[p])$ 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.

5.9. One approach to approximating a real-valued function with no closed-form expression over an interval is to (1) evaluate the function at $n$ equally-spaced points and (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, $n$ may be increased until the root mean squared error is tolerably close to zero.

Is this approach sensible? If not, what method would you recommend? Justify your response.

5.10. 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 $\sigma$. 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.

5.11. A standard biological model for predator-prey interactions, known as the Lokta-Volterra model, can be written

$$x' = \alpha x - xy$$

$$y' = xy - y,$$

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 $\alpha$ when there are no predators present and the greater the number of predators, the more slowly the prey population grows and it declines when the predator population exceeds $\alpha$. 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].

5.12. A frequently used model in finance for pricing bonds and futures (the so-called affine diffusion model) requires solving a system of Riccati (quadratic) differential equations of the form

$$\frac{dX}{dt} = A^\top X + \tfrac{1}{2} B^\top \mathrm{diag}(C^\top X) C^\top X - g$$

$$\frac{dx}{dt} = a^\top X + \tfrac{1}{2} b^\top \mathrm{diag}(C^\top X) C^\top X - g_0$$

where $X(t) : \Re_+ \to R^n$ and $x(t) : \Re_+ \to R$. The problem parameters $a$, $b$, and $g$ are $n \times 1$, $A$, $B$, and $C$ are $n \times n$ and $g_0$ is a scalar. In addition, the functions must satisfy boundary conditions of the form $X(0) = X_0$ and $x(0) = x_0$.

a) Write a program to solve this class of problems with the following input/output syntax:

```
[X,x]=AffSolve(t,a,A,b,B,C,g,g0,X0,x0)
```

The solution should be computed at the time values specified by `t`. If there are $m$ time values the outputs should be $m \times n$ and $m \times 1$. The program may use `RK4` or one of the functions in MATLAB's ODE suite (`ODE45` is particularly useful for this problem). You will need to write an auxiliary function to pass to the solver. Also note that $\mathrm{diag}(z)z$ can be written in MATLAB as `z.*z`.

Plot your solution functions over the interval $t \in [0, 30]$ for the following pa-

rameters values:

$$a = \begin{bmatrix} 0.0217 \\ 0.0124 \\ 0.00548 \end{bmatrix} \quad A = \begin{bmatrix} -17.4 & 17.4 & -9.309 \\ 0 & -0.226 & 0.879 \\ 0 & 0 & -0.362 \end{bmatrix}$$
$$b = \begin{bmatrix} 0 \\ .0002 \\ 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & .00782 \end{bmatrix}$$
$$g = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad C = \begin{bmatrix} 1 & -3.42 & 4.27 \\ -.0943 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

with $g_0 = x_0 = 0$ and $X_0 = 0$.

b) When the eigenvalues of $A$ are all negative (or have negative real parts when complex), $X$ has a long-run stationary point.

Write a fixed-point algorithm to compute the long-run stationary value of $X$, noting that it satisfies $dX/dt = 0$, testing it with the parameter values above. You should find that

$$X(\infty) = \begin{bmatrix} -0.0575 \\ -4.4248 \\ -8.2989 \end{bmatrix}.$$

Also write a a stand-alone algorithm implementing Newton's method for this problem (it should not call other functions like `newton` or `fjac`). To calculate the relevant Jacobian, it helps to note that

$$\frac{dAz}{dz} = A$$

and

$$\frac{d\text{diag}(z)z}{dz} = 2\text{diag}(z).$$

5.13. Show that the absolute value of the $O(h^3)$ truncation error in the second order Runge-Kutta formula (5.1):

$$x(t+h) = x + h\Big[\left(1 - \frac{1}{2\lambda}\right) f + \frac{1}{2\lambda} f\Big(t + \lambda h, x + \lambda h f\Big)\Big] + O(h^3).$$

is minimized by setting $\lambda = 2/3$.
(Hint: expand to the 4th order and minimize the $O(h^3)$ term.)

## Bibliographic Notes

Treatments of numerical integration are contained in most general numerical analysis texts. Press et al. contains a excellent treatment of Gaussian quadrature techniques. Our discussion of quasi-Monte Carlo techniques largely follows that of Judd.

A detailed treatment of the issues in computing finite difference approximations to derivatives is contained in Gill et al. (especially Section 8.6).

The subject of solving initial value problems is one the most studied in numerical analysis. See discussions, for example, in Atkinson, Press et al., and Golub and Ortega. MATLAB has a whole suite of ODE solvers, of which `ODE45` and `ODE15s` are good for most problems. `ODE15s` is useful for stiff problems and can also handle the slightly more general problem:

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

which includes $M$, the so-called mass matrix. We will encounter (potentially) stiff problems with mass matrices in Section ??.

The commercial fishery example was developed by Smith.

# 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 knows the value of a function $f$ at specified points in its domain and must choose an approximant $\hat{f}$ from a family of "nice", tractable functions that matches the original function at the known evaluation points. The interpolation problem can be generalized to include the value of the function's first or higher derivatives at specified 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

$$Tf = 0$$

where $T$ is an operator that maps a vector space of functions into itself. In the equivalent *functional fixed-point* problem, one must find a function $f$ such that

$$f = Tf.$$

The operator notation encompasses many specific cases, including simple functional relationships $g(x, f(x)) = 0$, differential equations $g(x, f(x), f'(x)) = 0$ and integral equations $f(x) - \int g(x)f(x)dx = 0$. In each of these examples, $g$ is a known function and the unknown function $f$ must satisfy the relationship for every value of $x$ on some specified domain.

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. Euler equations and the differential equations arising in arbitrage-based asset pricing models 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 explicit 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 of interpolation methods.

In this chapter we discuss methods for approximating functions and focus on the two most generally practical techniques: Chebychev polynomial and polynomial spline approximation. In addition we discuss the use of piecewise linear functions with finite difference approximations for derivatives. 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. Collocation will be used extensively in Chapters 9 and 11 for solving dynamic economic models.

In this chapter we also introduce a set of MATLAB functions that will be used extensively in the remainder of the book. These are discussed in Section 6.7. The suite of functions for working with approximating functions provide the foundation for the solution methods used for solving dynamic models in Chapters 9 and 11. The general boundary value solver discussed in this chapter (page 164) introduces the approach we will take.

## 6.1 Interpolation Principles

Interpolation involves the use of an approximating function, $\hat{f}$, that is easy to evaluate in place of the function of interest, $f$. The first step in designing an interpolation scheme is choose a family of approximating functions. We will confine ourselves to families of functions that can be written as a linear combination of a set of $n$ linearly independent *basis functions* $\phi_1, \phi_2, \ldots, \phi_n$:

$$\hat{f}(x) = \sum_{j=1}^{n} \phi_j(x) c_j = \phi(x) c,$$

whose *basis coefficients* $c_1, c_2, \ldots, c_n$ are to be determined.[1] Polynomials of increasing order are often used as basis functions, although other types of basis functions, most notably spline functions, are also common. 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*

$$\sum_{j=1}^{n} \phi_j(x_i) c_j = f(x_i) = y_i \qquad \forall i = 1, 2, \ldots, n.$$

Using matrix notation, the interpolation conditions equivalently may be written as the matrix linear *interpolation equation* whose unknown is the vector of basis coefficients $c$:

$$\Phi c = y,$$

where

$$\Phi_{ij} = \phi_j(x_i)$$

is the typical element of the *interpolation matrix* $\Phi$. 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. 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 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

$$\sum_{j=1}^{n} \phi_j(x_i) c_j \quad = \quad f(x_i), \qquad \forall i = 1, \ldots, n_1$$

[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 except in approximating special functions such as cumulative distribution functions.

$$\sum_{j=1}^{n} \phi_j'(x_i')c_j \quad = \quad f'(x_i'), \qquad \forall i = 1, \ldots, n_2$$

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.

Interpolation is closely related to the problem of curve fitting; indeed it can be thought of as a special case. The curve fitting problem arises when one attempts to find an approximant that has lower degree than the number of available evaluation points. In this case it will not generally be possible to solve

$$\sum_{j=1}^{n} \phi_j(x_i)c_j = f(x_i)$$

exactly. Instead one can define an approximation error

$$e_i = f(x_i) - \sum_{j=1}^{n} \phi_j(x_i)c_j,$$

and attempt to make the norm of $e$ small. If the 2-norm is used, this is the least squares curve fitting problem:

$$\min_c \sum_{i=1}^{m} e_i^2.$$

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 of 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 low 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, for any $\epsilon > 0$, there exists a polynomial $p$ such that

$$||f - p||_\infty = \sup_{x \in [a,b]} |f(x) - p(x)| < \epsilon.$$

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 polynomial approximants with evenly spaced nodes 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 for $n = 9$, 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.

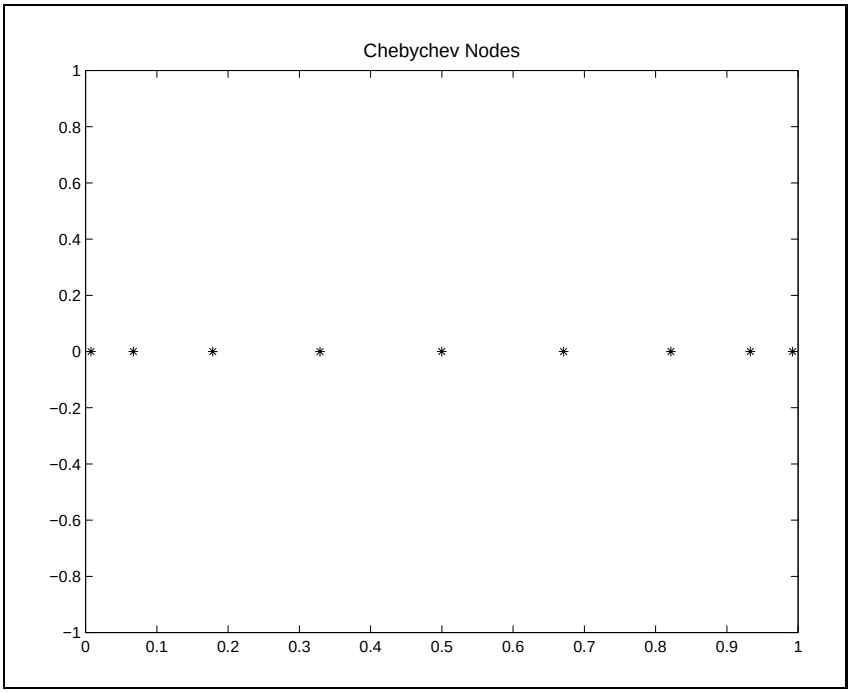


Figure 6.1

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 approximation error associated with the $n^{th}$-degree Chebychev-node polynomial interpolant cannot be 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, provides 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 in-

terpolant. More importantly, however, the error bound goes to zero as $n$ rises. In contrast to polynomial interpolation with evenly spaced nodes, 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 such endpoint instabilities because the nodes are more heavily concentrated near the endpoints.

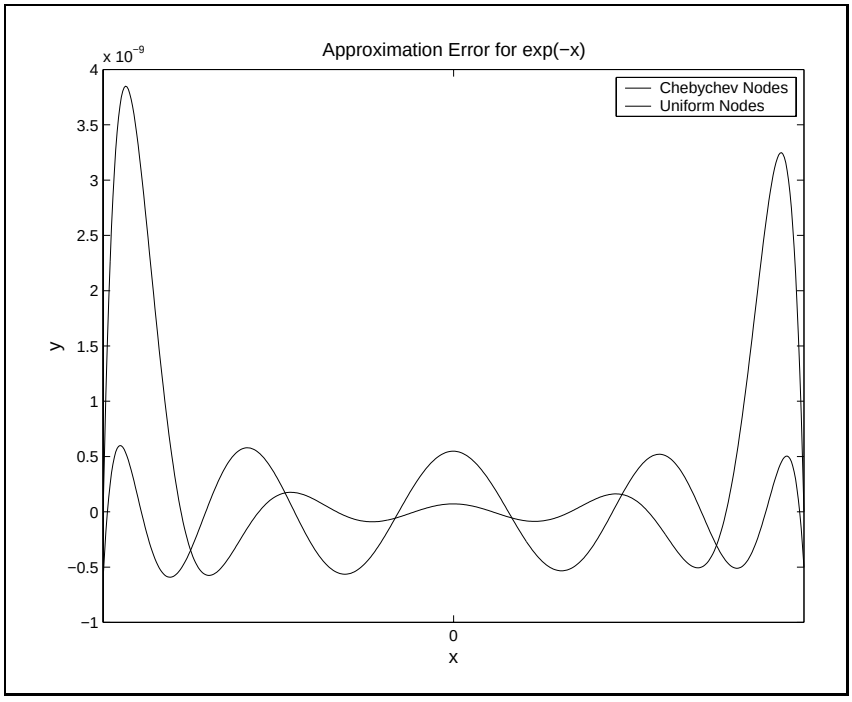


Figure 6.2

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 for the interval $x \in [0, 1]$. However, the monomial basis produces an interpolation matrix $\Phi$ that is a so-called Vandermonde

matrix:

$$\Phi = \begin{bmatrix} 1 & x_1 & \dots & x_1^{n-2} & x_1^{n-1} \\ 1 & x_2 & \dots & x_2^{n-2} & x_2^{n-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & x_n & \dots & x_n^{n-2} & x_n^{n-1} \end{bmatrix}.$$

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.

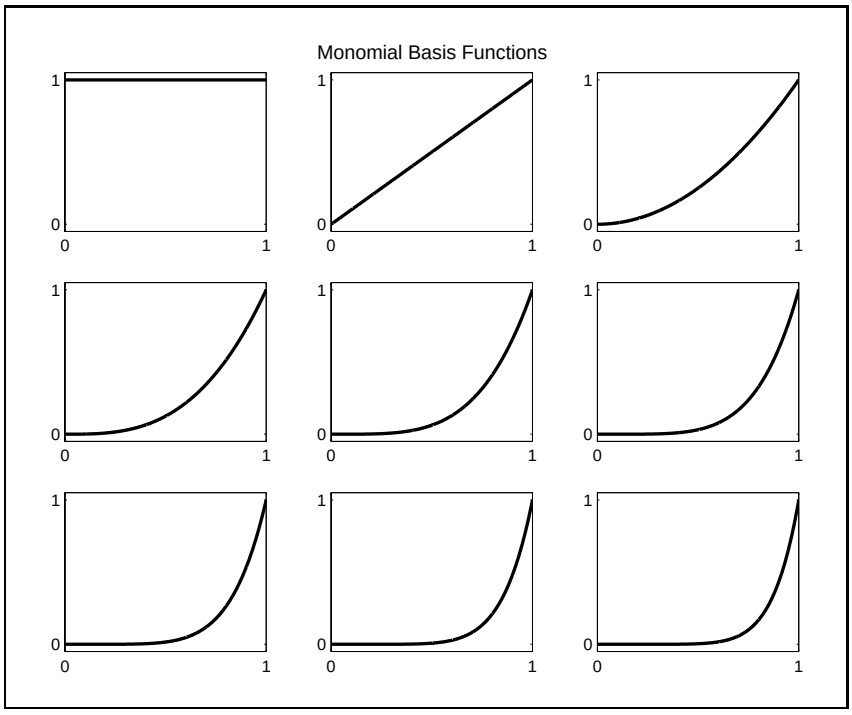


Figure 6.3

Fortunately, alternatives to the standard monomial basis exist. In fact, any sequence of $n$ polynomials having exact orders $0, 1, 2, \dots, n-1$ 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. Defining $z = 2(x-a)/(b-a) - 1$, to normalize the domain to the interval [-1,1], the Chebychev polynomials are defined

recursively as:[2]

$$\phi_j(x) = T_{j-1}(z)$$

where

$$\begin{aligned} T_0(z) &= 1 \\ T_1(z) &= z \\ T_2(z) &= 2z^2 - 1 \\ T_3(z) &= 4z^3 - 3z \\ &\vdots \\ T_j(z) &= 2zT_{j-1}(z) - T_{j-2}(z). \end{aligned}$$

The first twelve Chebychev basis polynomials for the interval $x \in [0, 1]$ are displayed in Figure 6.4.

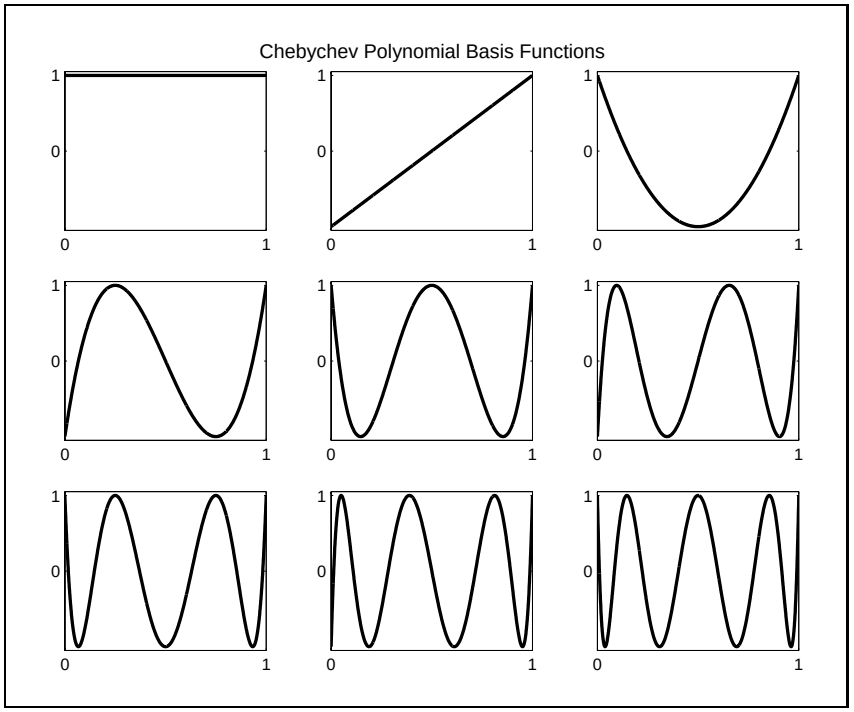


Figure 6.4

[2]The Chebychev polynomials also possess the alternate trigonometric definition $T_j(z) = \cos\Big(\arccos(z)j\Big)$ on the domain $[a, b]$.

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 with high degree approximants. The interpolation matrix $\Phi$ 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

$$\Phi^\top \Phi = \text{diag}\{n, n/2, n/2, \ldots, n/2\}$$

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.

Derivatives and integrals of polynomials are also polynomials. Differentiation decreases the polynomial order by 1 and integration increases it by 1. A differential operator that maps the coefficients of a polynomial in the Chebyshev basis is given by the $n - 1 \times n$ matrix operator with $ij$th element given by

$$D_{ij} = \begin{cases} \frac{2(j-1)}{b-a} & \text{if } i = 1 \text{ and } i + j \text{ is odd} \\ \frac{4(j-1)}{b-a} & \text{if } i > 1,\ i + j \text{ is odd and i¡j} \\ 0 & \text{otherwise.} \end{cases}$$

Similarly, the $n + 1 \times n$ matrix with $ij$th element

$$D_{ij}^{-1} = \begin{cases} \frac{b-a}{2} & \text{if } i \leq 2 \text{ and } j = 1 \\ -\frac{b-a}{8} & \text{if } i = 1 \text{ and } j = 2 \\ \frac{(-1)^j (b-a)}{2j(j-2)} & \text{if } i = 1 \text{ and } j > 2 \\ \frac{b-a}{4j} & \text{if } i > 1 \text{ and } j = i - 1 \\ -\frac{b-a}{4(j-2)} & \text{if } i > 1 \text{ and } j = i + 1 \\ 0 & \text{otherwise.} \end{cases}$$

maps Chebyshev coefficients into the coefficients of the integral (normalized so the integral is 0 at $a$).

## 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, an order $k$ spline consists of series of $k^{th}$ order 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, an order $k$ 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 order $k$ 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 $k$ spline with breakpoint vector $\nu$ 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, $\nu$, 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 as the sum of B-splines of order $k+1$. Thus, the antiderivative of an order $k$ spline is an order $k+1$ spline with the same breakpoints. This implies that the family of splines are closed under differentiation and integration, with the oder $k$ decreasing or increasing by 1 and with the breakpoints remaining unchanged.

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

$$\hat{f}(x) = \sum_{i=1}^{n} \phi_i(x) c_i$$

of the basis functions:

$$\phi_j(x) = \begin{cases} 1 - \dfrac{|x - \nu_j|}{h} & \text{if } |x - \nu_j| \leq h \\ 0 & \text{otherwise.} \end{cases}$$

Here, $h = (b-a)/(n-1)$ is the distance between breakpoints and $\nu_j = a + (j-1)h$, $j = 1, 2, \ldots, n$, 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 functions for a degree 12, evenly spaced breakpoint linear spline on the interval $[0, 1]$. Each hat function is zero everywhere, except over a narrow support of width $2h$. The basis function achieves a maximum of 1 at the midpoint of its support.

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

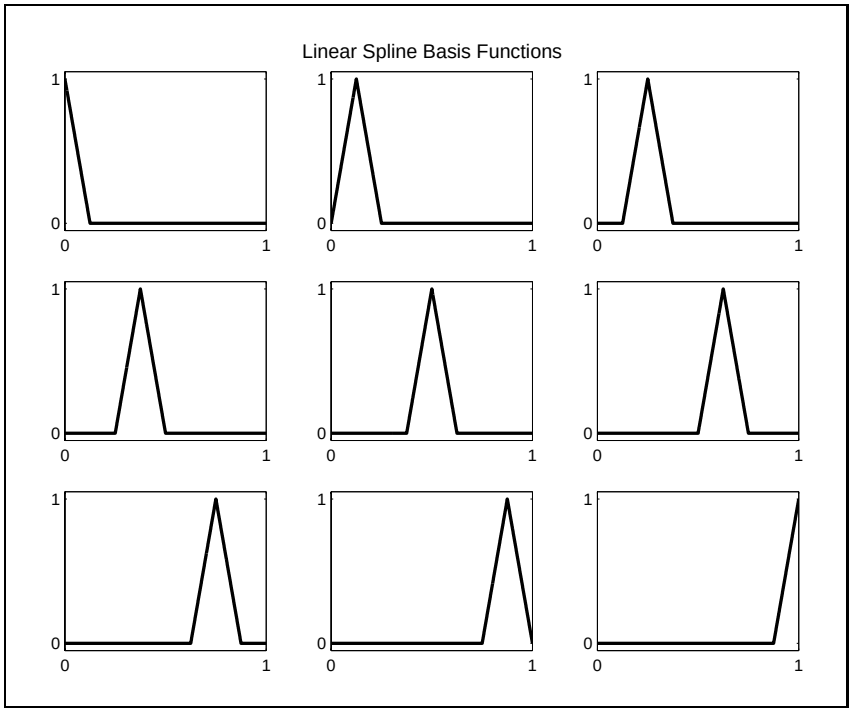


Figure 6.5

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. In this case $\phi_i(x_j)$ equals one if $i = j$, but equals zero otherwise; that is, the interpolation matrix $\Phi$ is simply the identity matrix and the interpolation equation reduces to the identity $c = y$, where $y$ is the vector of function values at the interpolation nodes. The linear spline approximant of $f$ when nodes and breakpoints coincide thus takes the form

$$\hat{f}(x) = \sum_{i=1}^{n} \phi_i(x) f(x_i).$$

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 $1 + (x - a)/h$, then $x$ lies in the interval $[\nu_i, \nu_{i+1}]$. Thus,

$$\hat{f}(x) = ((x - \nu_i)c_{i+1} + (\nu_{i+1} - x)c_i)/h$$

and

$$\hat{f}'(x) = (c_{i+1} - c_i)/h.$$

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. The first and maybe second derivatives of the function also may be needed to solve for the root of the function using Newton-like method and in the continuous time dynamic models encountered in Chapters 10 and 11 are expressed in terms of second order differential equations.

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 $\nu_j = a + h(j - 1)$, $j = 1, 2, \ldots, n - 2$, where $h = \frac{b-a}{n-3}$. Cubic spline basis function generated with evenly spaced breakpoints are nonzero over a support of width $4h$. As such, at any point of $[a, b]$, at most four basis functions are nonzero. The basis functions for a degree 12, evenly spaced breakpoint cubic spline on the interval $[0, 1]$ are illustrated in Figure 6.6.

Although spline breakpoints are often chosen to be evenly spaced, 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 polynomials of high degree. With splines, however, one can often finesse curvature difficulties by concentrating breakpoints in regions displaying the highest degree of curvature.

To illustrate the importance of breakpoint location, consider the problem of forming a cubic spline approximant for Runge's function

$$f(x) = \frac{1}{1 + 25x^2}$$

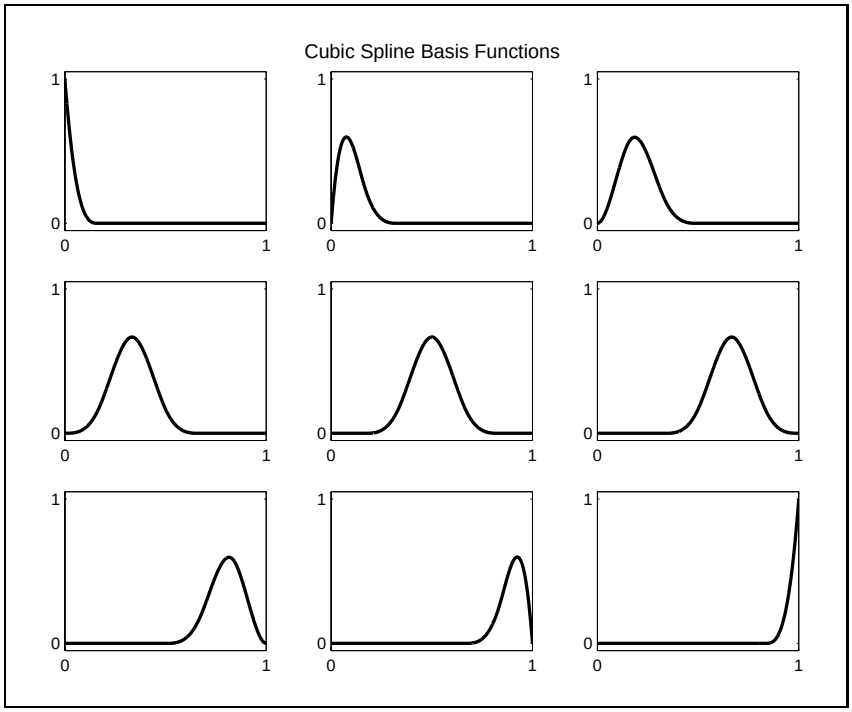


Figure 6.6

on the interval $x \in [-5, 5]$. 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 demonstrate 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 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

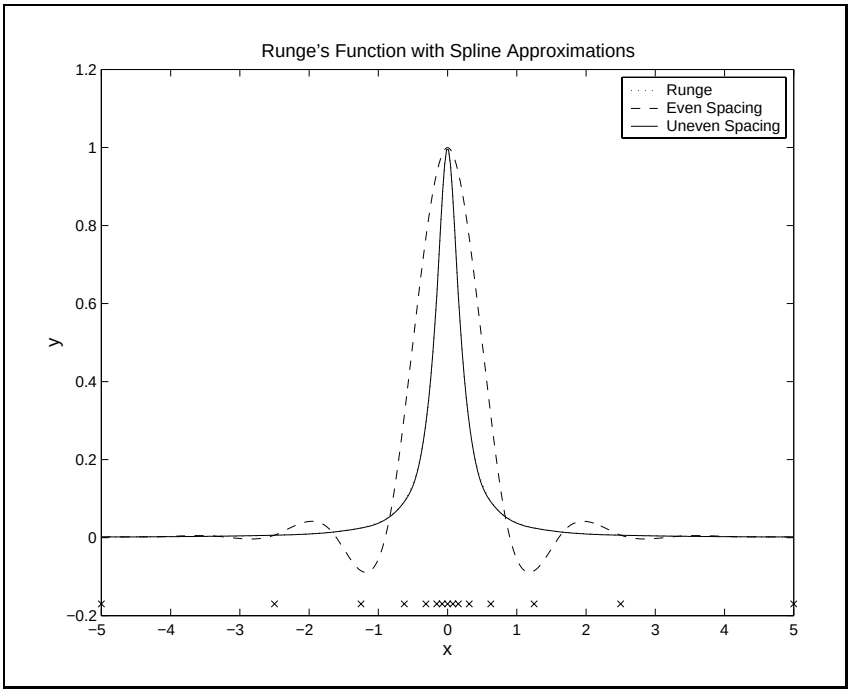


Figure 6.7

given point. Kinks arise in the pricing of options, which display a kink in their payoff function and in dynamic optimization problems with discrete choice variables, which display kinks in their marginal value function (or its derivative).

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 using sparse matrix methods. 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 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

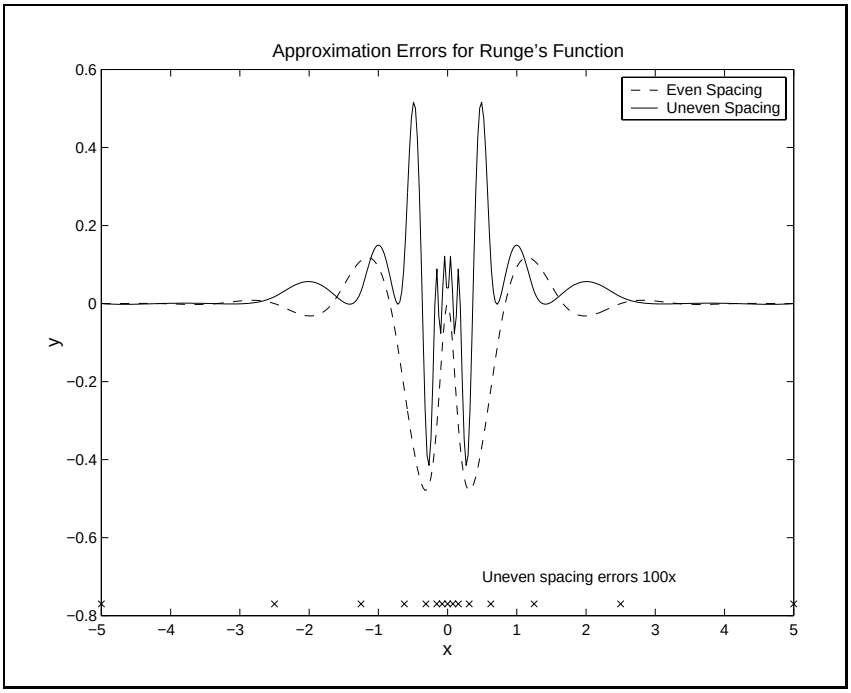


Figure 6.8

dense inverse interpolation matrix (accomplished with the "\" operator in MATLAB).

## 6.4 Piecewise-Linear Basis Functions

Despite their simplicity, linear splines have many virtues. For problems in which the function being approximated is not-smooth and may even exhibit discontinuities, linear splines can still provide reasonable approximations. Unfortunately, derivatives of linear splines are discontinuous, piecewise constant functions.

There is no reason, however, to limit ourselves to using the actual derivative of the approximating function, if a more suitable alternative exists. If a function is approximated by a linear spline, a reasonable candidate for an approximation of its derivative is a linear spline constructed using finite difference approximations to the derivative (see Section 5.6, page 107). Given a breakpoint sequence $\nu$ for the function's approximant, this can be accomplished by defining a new breakpoint sequence with $n-1$ values placed at the midpoints of the original sequence: $z_i = (\nu_i + \nu_{i+1})/2$, $i = 1, \ldots, n-1$. The new function is set to equal the centered finite difference

approximation to the derivative at the new breakpoints:

$$f'(z_i) \approx \frac{f(\nu_{i+1}) - f(\nu_i)}{\nu_{i+1} - \nu_i}.$$

Values between and beyond the $z_i$ sequence can be obtained by linear interpolation and extrapolation. We leave it as an exercise to show that this piecewise linear function, evaluated at the original breakpoints (the $\nu_i$), is equal to the centered finite difference approximations derived in the last chapter. Approximations to higher order derivatives can be obtained be repeated application of this idea.

For completeness, we define an approximate integral that is also a linear spline, with a breakpoint sequence $z_{i+1} = (\nu_i + \nu_{i+1})/2$ for $i = 2, \ldots, n$ and with additional breakpoints defined by extrapolating beyond the original sequence: $z_1 = (3\nu_1 - \nu_2)/2$ and $z_{n+1} = (3\nu_n - \nu_{n-1})/2$. The approximation to the integral,

$$F(x) = \int_{\nu_1}^{x} f(x)dx$$

at the new breakpoints is

$$F(z_i) = F(z_{i-1}) + (z_i - z_{i-1})f(\nu_{i-1}),$$

where

$$F(z_1) = \tfrac{1}{2}(\nu_1 - \nu_2)f(\nu_1)$$

(this ensures the normalization that $F(\nu_1) = 0$).[3] This definition produces an approximation to the integral at the original breakpoints that is equal to the approximation obtained by applying the trapezoid rule (see Section 5.1, page 95):

$$\int_{\nu_i}^{\nu_{i+1}} f(x)dx \approx \tfrac{1}{2}(\nu_{i+1} - \nu_i)(f(\nu_{i+1}) + f(\nu_i)).$$

(we leave the verification of this assertion as exercise for the reader).

As with the other families of functions discussed, the family of piecewise linear functions obtained using these approximations is closed under differentiation and integration. Unlike splines, however, for which differentiation and integration decreases or increases the order of the piecewise segments, leaving the breakpoint sequence

[3]It should be pointed out that the breakpoint sequence obtain by integrating and then differentiating will not produce the original breakpoint sequence unless the original breakpoints are evenly spaced. This leads to the unfortunate property that differentiating the integral will only produce the original function if the breakpoints are evenly spaced. It can also be shown that, although the first derivatives are $O(h^2)$, the second derivatives are only $O(h)$ when the breakpoints are not evenly spaced.

unchanged, with the piecewise linear family, differentiation and integration do not change the polynomial order of the pieces (they remain linear) but decrease or increase the number of breakpoints.

The piecewise linear family makes computation using finite difference operators quite easy, without a need for special treatment to distinguish them from other families of basis functions (including finite element families such as splines). We will return to this point in Chapter 11 when we discuss solving partial differential equations (PDEs).

## 6.5 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 $\phi_i^x$, $i = 1, 2, \ldots, n_x$ and $\phi_j^y$, $j = 1, 2, \ldots, n_y$ 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

$$\hat{f}(x, y) = \sum_{i=1}^{4} \sum_{j=1}^{3} x^{i-1} y^{j-1} c_{ij}.$$

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

$$I = \{(x_1, x_2, \ldots, x_d) \mid a_i \leq x_i \leq b_i, i = 1, 2, \ldots, d\}.$$

Let $\phi_{ij}$, $j = 1, ..., n_i$, be the $j$th basis function in a $n_i$ degree univariate basis for real-valued functions of on $[a_i, b_i]$. An approximant for $f$ in the tensor product basis would take the following form:

$$\hat{f}(x_1, x_2, \ldots, x_d) = \sum_{j_1=1}^{n_1} \sum_{j_2=1}^{n_2} \cdots \sum_{j_d=1}^{n_d} \phi_{1j_1}(x_1)\phi_{2j_2}(x_2) \ldots \phi_{dj_d}(x_d)c_{j_1 \ldots j_d}.$$

Using tensor notation the approximating function can be written

$$\hat{f}(x_1, x_2, \ldots, x_d) = [\phi_d(x_d) \otimes \phi_{d-1}(x_{d-1}) \otimes \ldots \otimes \phi_1(x_1)]c.$$

where $c$ is a column vector with $n = \Pi_{i=1}^{d} n_i$ elements. We have chosen to evaluate the tensor product in reverse order; in principle it can be evaluated in any order but using the reverse order makes indexing easier in MATLAB. 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

$$\phi_1(x_1) = [1 \; x_1 \; x_1^2]$$

and

$$\phi_2(x_2) = [1 \; x_2].$$

Finally, the full basis function is

$$\phi(x) = [1 \; x_2] \otimes [1 \; x_1 \; x_1^2] = [1 \; x_1 \; x_1^2 \; x_2 \; x_1 x_2 \; x_1^2 x_2],$$

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 $\phi(X_i)$. 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

$$\Phi(X) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0.5 & 0 & 0 \\ 1 & 0.5 & 0.25 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix},$$

which is $4 \times 6$ ($m \times n$).

If $\Phi(X)$ has $n$ rows and is nonsingular, the coefficients of the interpolating function are found by performing the linear solve

$$\Phi(X)c = f(X), \tag{6.1}$$

where $f(X)$ represents the $m$ values of the function evaluated at each of the $X_i$. This is one of the reasons why we have limited ourselves to families of functions that can be expressed as linear combinations of basis functions; it is easy to solve the interpolation problem.

Although (6.1) can be solved for arbitrary values of an $n \times d$ matrix $X$ as long as the resulting $\Phi(X)$ is nonsingular, substantial efficiencies can be obtained if $X$ represents points on a regular grid. Specifically, suppose we form the grid defined by the $d$ vectors $x_i$, the $i$th of which has $n_i$ values. If $\Phi_i$ is the $n_i \times n_i$ interpolation matrix

associated with $x_i$, then the interpolation conditions for the multivariate function can be written

$$[\Phi_d \otimes \Phi_{d-1} \otimes \ldots \otimes \Phi_1]c = f(X)$$

where $f(X)$ contains the $n$ values of the function evaluated at the interpolation nodes $x$, properly stacked.

As an example, suppose $x_1 = [0; 0.5; 1]$ and $x_2 = [0; 1]$ and we use the monomial basis functions of the above example. Then

$$\Phi_1(X_1) = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0.5 & 0.25 \\ 1 & 1 & 1 \end{bmatrix},$$

$$\Phi_2(X_2) = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

and

$$\Phi(X) = \Phi_2(X_2) \otimes \Phi_1(X_1) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0.5 & 0.25 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0.5 & 0.25 & 1 & 0.5 & 0.25 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

The proper stacking of $X$ yields rows containing all possible combinations of the values of the $x_i$, with the lowest order $x_i$ changing most rapidly:[4]

$$X = \begin{bmatrix} 0 & 0 \\ 0.5 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0.5 & 1 \\ 1 & 1 \end{bmatrix}.$$

Using a standard result from tensor matrix algebra, the system can be solved by forming the inverse of the interpolation matrix and postmultiplying it by the data vector:

$$c = [\Phi_d^{-1} \otimes \Phi_{d-1}^{-1} \otimes \ldots \otimes \Phi_1^{-1}]f(X).$$

[4] If we formed the tensor products in ascending rather than descending order, we should have the highest order $x_i$ changing most rapidly; this, however, runs counter to MATLAB's indexing conventions.

Hence, there is no need to invert an $n \times n$ 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 there are 10 evaluation points in each dimension, only three $10 \times 10$ matrices need to be inverted, rather than a single $1000 \times 1000$ 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*. 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.

## 6.6 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 approximants obtained using splines and Chebyshev polynomials, along with the actual functions, are

displayed in Figures 6.9-6.12.

The four functions are ordered in increasing difficulty of approximation. The first is cubic 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 and in Figures 6.9-6.12 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; otherwise, cubic or even linear spline interpolation may be preferred.

## 6.7 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.

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

**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 |
| $\lvert x\rvert^{0.5}$ | 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 |

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:

```
fspace = fundefn(bastype,n,a,b,order);
```

Here, on input, `bastype` is string referencing the basis function family, which can take the values `'cheb'` for Chebychev polynomial basis, `'spli'` for spline basis or `'lin'` for piecewise linear 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 (only used if `bastype` is `'spli'`). On output, `fspace` 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 10th degree Chebychev approximants for univariate functions defined on the interval $[-1, 2]$. Then one would define the appropriate function space for approximation as follows:

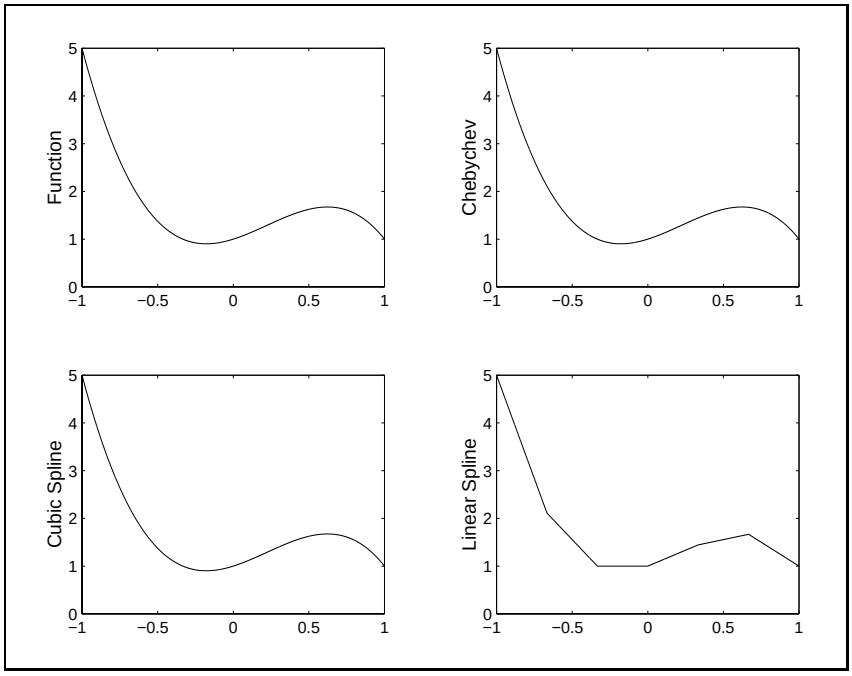


Figure 6.9

```
fspace = fundefn('cheb',10,-1,2);
```

Suppose instead 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 10 basis functions for the $x_1$ dimension and 15 basis functions for the $x_2$ dimension. Then one would issue the following command:

```
fspace = 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. The syntax for this function approximation routine is:

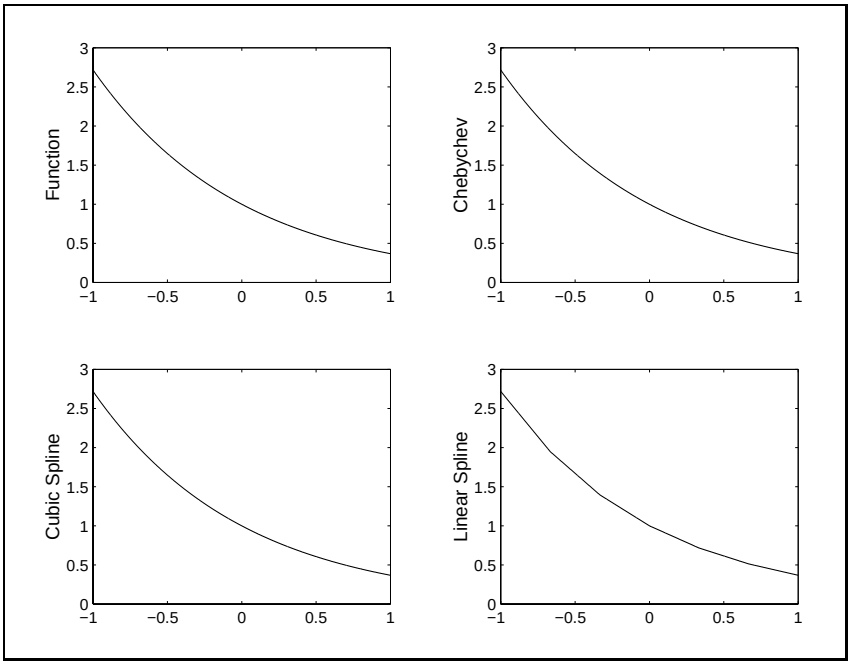


Figure 6.10

```
c = funfitf(fspace,f,additional parameters);
```

Here, on input, `fspace` is the approximation function space defined using `FUNDEF`, `f` is the string name of the m-file that evaluates the function to be approximated. Any additional parameters passed to `FUNFITF` are simply 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(fspace,x,y);
```

Here, on input, `fspace` is an approximation function space defined using `FUNDEF`, `x` is a matrix of points at which the function has been evaluated (each row represents one point in $R^d$) and `y` is a matrix of function values at those points. On output, `c` is the matrix 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

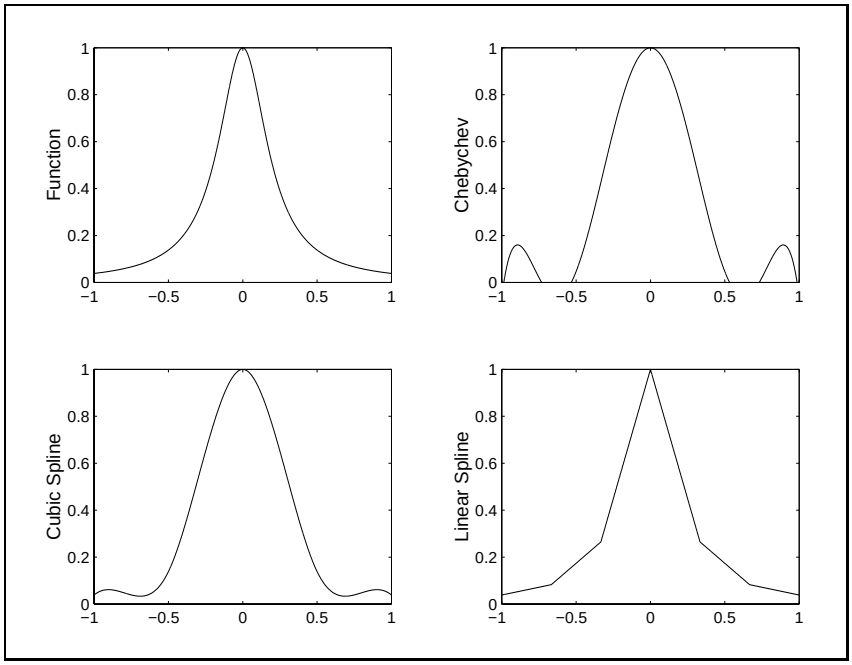


Figure 6.11

data points than coefficients, `FUNFITXY` returns the least squares fit; the procedure can therefore be used for statistical data fitting as well as interpolation.

If `y` is obtained by evaluating $f$ at a regular grid of values, $x$ can be passed as a cell array containing the vectors defining the grid. The toolbox contains a function, `makegrid`, for generating grid points from such a cell array. To evaluate a function on a regular grid one can use the following code:

```
X=makegrid(x);
y=f(X);
```

If `x` a cell array containing $d$ vectors of length $n_i$, $i = 1, \ldots, d$, `makegrid` returns `X` as an $\prod_i n_i \times d$ matrix. One could then use either of the following commands to obtain the coefficient values:

```
c=funfitxy(fspace,x,y);
```

or

```
c=funfitxy(fspace,X,y);
```

The only difference in calling syntax involves whether `x` or `X` is passed. Using `x` however is far more efficient when $d > 1$ because the interpolation equation can be

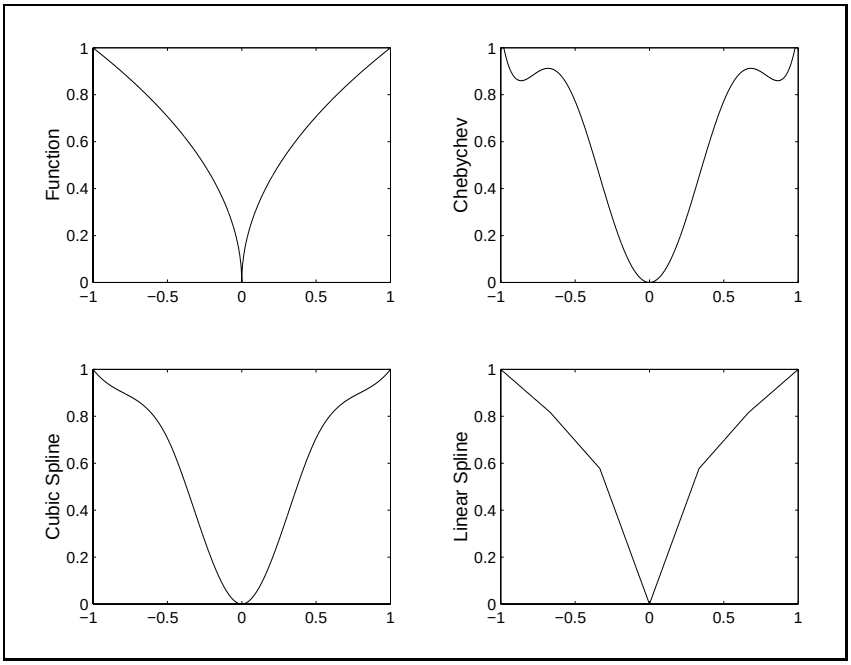


Figure 6.12

solved using the tensor product of the inverses rather than the inverse of the tensor product.

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,fspace,x);
```

Here, on input, `fspace` is the approximation function space defined using `FUNDEFN`, `c` is the matrix of coefficients that identifies the approximant and `x` is the point at which the approximant is to be evaluated, written as a $m \times d$ matrix. 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 $m \times d$ array `x`, in which case `y` is returned as an $m \times 1$ 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 $1 \times d$ 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]);
```

The single command

```
J = funeval(c,space,x,eye(d));
```

will compute the entire Jacobian. 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]);
f12 = funeval(c,space,x,[1 1]);
f22 = funeval(c,space,x,[0 2]);
```

A simple example 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

$$f(x) = \exp(-\alpha x)$$

on [-1,1]. The first step is to create a file that computes the desired function:

```
function fx=nexp(x,alpha)
fx=exp(-alpha*x);
```

The file should be named `nexp`. The following script constructs the Chebychev approximant for $\alpha = 2$ and then plots the errors using a finer grid than used in interpolation:

```
alpha=2;
space = fundefn('cheb',10,-1,1);
c     = funfitf(space,'nexp',alpha);
x     = nodeunif(1001,-1,1);
yact  = f(x);
yapp  = funeval(c,space,x);
plot(x,yact-yapp);
```

The steps used here are to first initialize the parameter $\alpha$. Second, we use `FUNDEFN` to define the function space from which the approximant is to be drawn, in this case the space of degree 10 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 6.13.

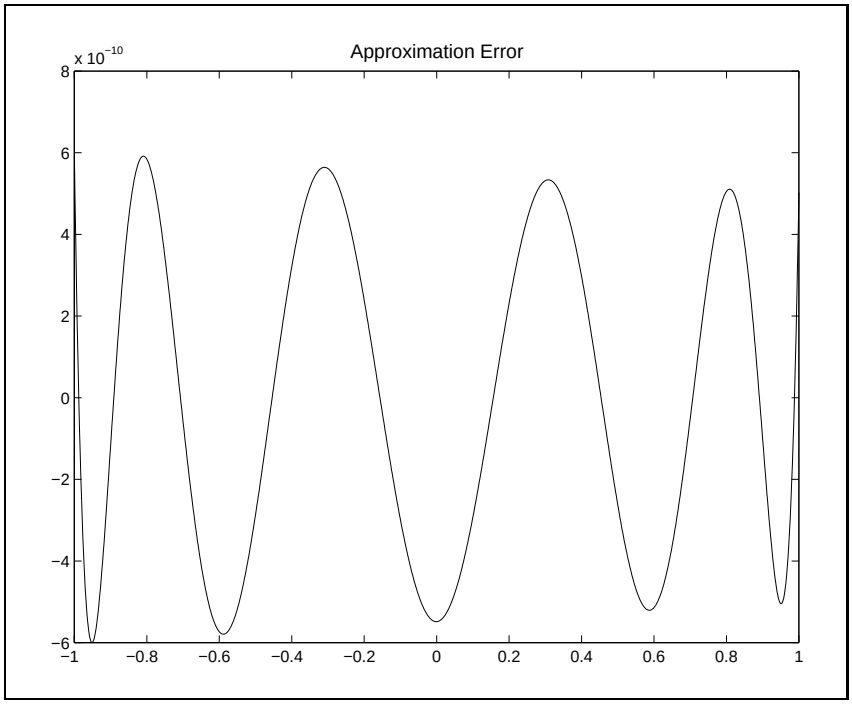


Figure 6.13

Two 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);
```

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. The commands

```
B = funbas(space,x);
y = B*c;
```

have the same effect as

```
y = funeval(c,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 function space. Its syntax is

```
x = funnode(space);
```

The toolbox also contains a number of functions for "power users." These functions either automate certain procedures (for example `FUNJAC`, `FUNHESS` and `FUNCONV`) or they give the user more control over how information is stored and manipulated (for example `FUNBASX` or `FUNDEF`). In addition, the function approximation tools are extensible, allowing other families of approximating functions to be defined. Complete documentation is available at the toolbox web site (see Web Resources on page 497).

## 6.8 Solving Functional Equations

In this section we consider the use of approximants to solve functional equations. One class of functional equation problems involves finding a function $f$ that satisfies

$$g(x, f(x)) = 0 \text{ for } x \in [a, b].$$

A numerical solution to this problem seeks a function $\hat{f}$ from a finite-dimensional function space that approximately satisfies $g(x, \hat{f}(x)) = 0$.

As with interpolation, it is useful to work with approximants that can be written in the form

$$f(x) \approx \hat{f}(x) = \sum_{j=1}^{n} \phi_j(x) c_j = \phi(x) c,$$

where the $\phi_j$ are a set of basis functions. The condition to be satisfied can be written as

$$g\left(x, \phi(x) c\right) \approx 0 \text{ for } x \in [a, b].$$

The term $g(x, \phi(x)c)$ can be thought of as a residual, which should be made small (in some sense) by the choice of $c$. 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:

$$g(x_i, \phi(x_i)c) = 0 \text{ for } i = 1, 2, \ldots, n.$$

This approach changes an infinite dimensional function equation problem into an $n$-dimensional rootfinding problem, which can be solved using the methods discussed in Chapter 3.

The same approach can be taken with respect to other classes of functional equations. Consider, for example, the differential equation

$$g(x, f(x), f'(x)) = 0 \text{ for } x \in [a, b]$$

subject to $b(f(x_b)) = 0$. This can be replaced by the residual function

$$g(x_i, \phi(x_i)c, \phi'(x_i)c) = 0 \text{ for } i = 1, 2, \ldots, n-1$$

and the boundary function

$$b(\phi(x_b)c) = 0.$$

Although the principle of collocation can be stated quite simply, it is a powerful tool in solving complicated economic equilibirum and optimization models. We now examine some examples of functional equations and demonstrate the use of collocation methods to solve them.

### 6.8.1 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[5]

$$q = S(p) = \eta(p - c)p^{-\eta-1}.$$

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

$$MC(q) = \alpha\sqrt{q} + q^2.$$

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 $\Phi$. 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:

[5] 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 $p < c$.

```
alpha=1; eta=1.5;
n=25; a=0.1; b=3;
fspace = fundefn('cheb',n,a,b);
p = funnode(fspace);
Phi = funbas(fspace,p);
c = Phi\sqrt(p);
c = broyden('fapp09',c,[],p,alpha,eta,Phi);
```

The script calls a function `'fapp09'` that computes the functional equation residual for any choice of coefficient vector $c$:

```
function resid=fapp09(c,p,alpha,eta,Phi);
dp    = (-1./eta)*p.^(eta+1);
q     = Phi*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 6.14. 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 6.15.

It should be emphasized that almost all collocation problems involve writing a function to compute the residuals which is passed to a root-finding algorithm (the most important exception is when the residual function is linear in $c$, which can therefore be computed using a linear solve operation). Typically, 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, for most problems, it is useful to write two procedures when solving collocation problems. The first sets up the problem and initializes variables. It then call a root-finding algorithm, passing it the name of the second procedure, which computes the residuals.

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:

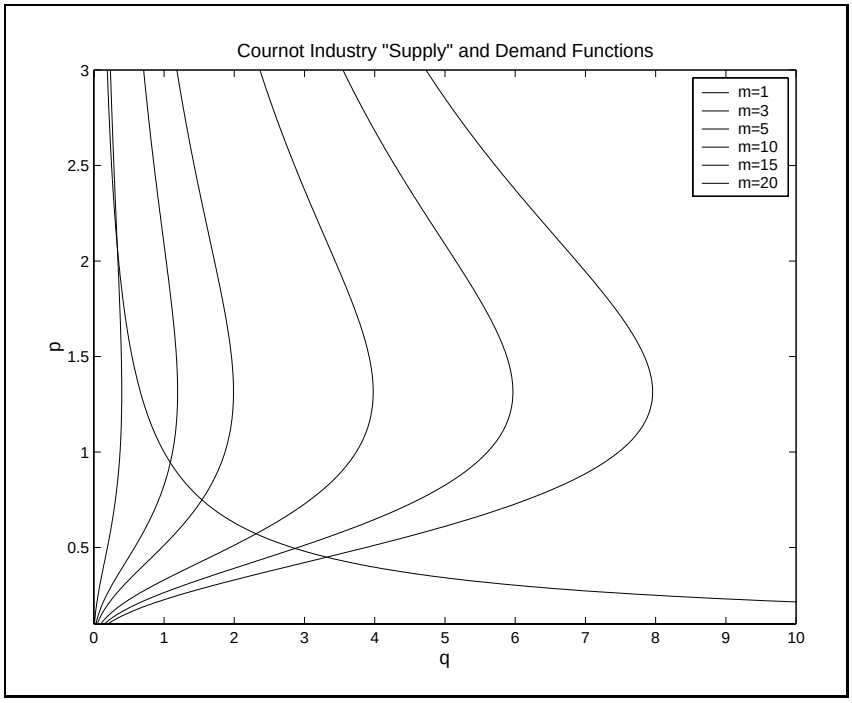


Figure 6.14

```
p = nodeunif(501,a,b);
Phi = funbas(fspace,p);
r = resid(c,p,alpha,eta,Phi);
plot(p,r)
```

The result is shown in Figure 6.16, which makes clear that the approximation adequately solves the functional equation (the file `demapp09` contains code for this example).

### 6.8.2 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

$$g\left(\phi(x_i)c\right) - x_i = 0$$

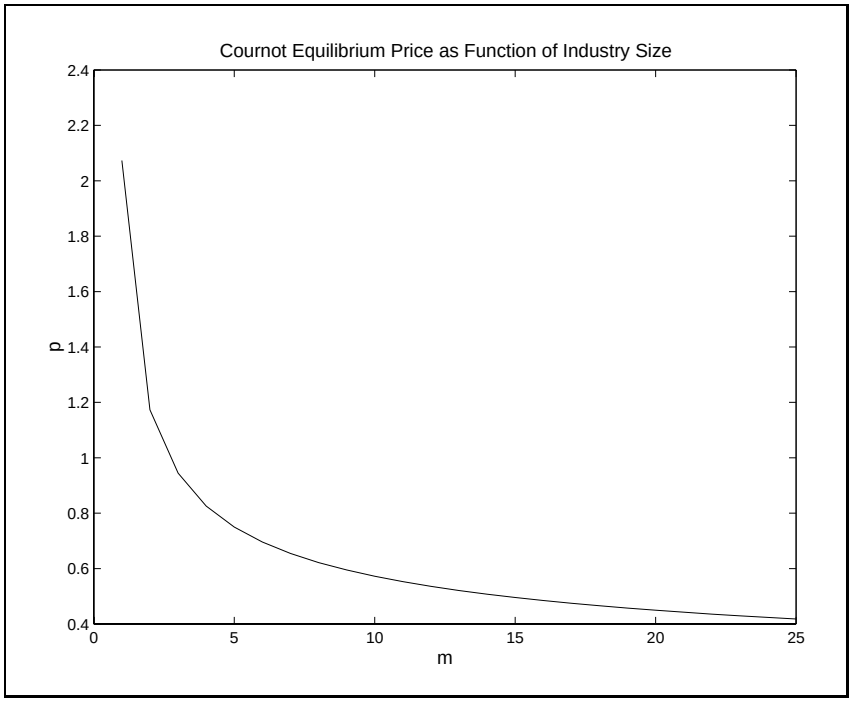


Figure 6.15

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.

To accomplish this we will first 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. It is also necessary to define initial coefficient values; we've simply defined an identity mapping, $f(x) = x$, as our initial guess. This works well 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 the range $x \in [1, 2]$. 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:

```
fspace = fundefn('cheb',6,1,2);           % define approximating family
x = funnode(fspace);                      % select collocation nodes
Phi = funbas(fspace,x);                   % define basis matrix
c = funfitxy(fspace,x,x);                 % initial conditions
c = broyden('fapp10',c,[],fspace,x,Phi);  % call solver
```

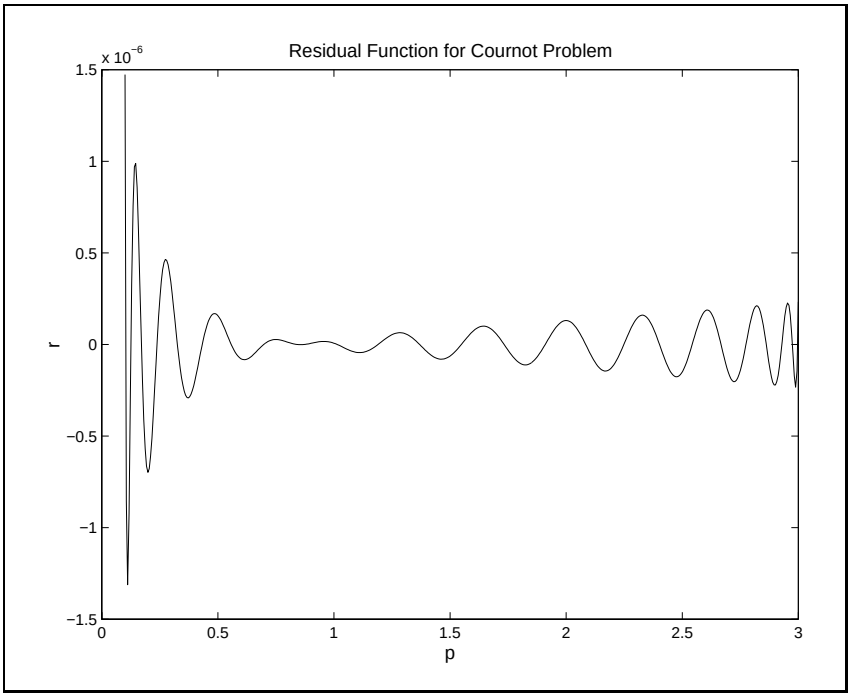


Figure 6.16

The script calls a function `'fapp10'` that computes the functional equation residual for any choice of coefficient vector $c$:

```
function resid=fapp10(c,fspace,x,Phi)
resid = exp(Phi*c)-x;
```

The script file `demapp10` demonstrates the method and generates a plot of the residual function, shown in Figure 6.17, and a plot of the true approximation error, shown in Figure 6.18. 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 directly how well we have done.

It would be a simple matter to write a general procedure that automated this process for finding function inverses; we leave this as an exercise.

### 6.8.3 Boundary Value Problems

Initial value problems of the form

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

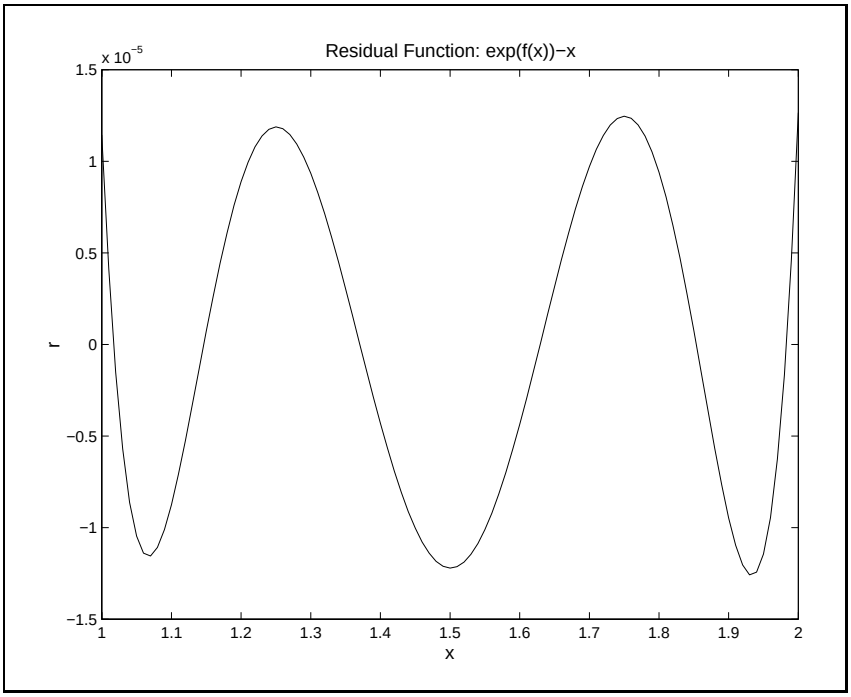


Figure 6.17

subject to $x(0) = x_0$ were discussed in Section 5.7 (page 114). A more general form of differential equations are the so-called boundary value problems (BVPs). In a BVP one seeks a solution function $x(t) : [a, b] \rightarrow R^d$ that satisfies the residual function

$$r(t, x(t), x'(t)) = 0$$

subject to $b_i(t_i^b, x(t_i^b), x'(t_i^b)) = 0$, for $i = 1, \ldots, d$. This generalizes the IVP in two ways. First, the differential equation can be non-linear in $x'(t)$. Second, the solution function need not be known at any specific point. Instead, $d$ side conditions of any form must be specified.

Boundary value problems arise most often in economics in deterministic optimal control problems, the solutions to which can be expressed as a set of differential equations defining the dynamic behavior of a set of $d_s$ state variables and $d_x$ control or decision variables, along with initial values for the $d_s$ state variables and $d_x$ so-called transversality conditions. We will consider such problems in more detail in Chapter 10.

Although there are a number of strategies to solve BVPs, the function approximation tools developed in this chapter make a collocation strategy very straightforward. The solution is approximated using $x(t) \approx \phi(t)c$, where $\phi$ is a set of $n$ basis functions

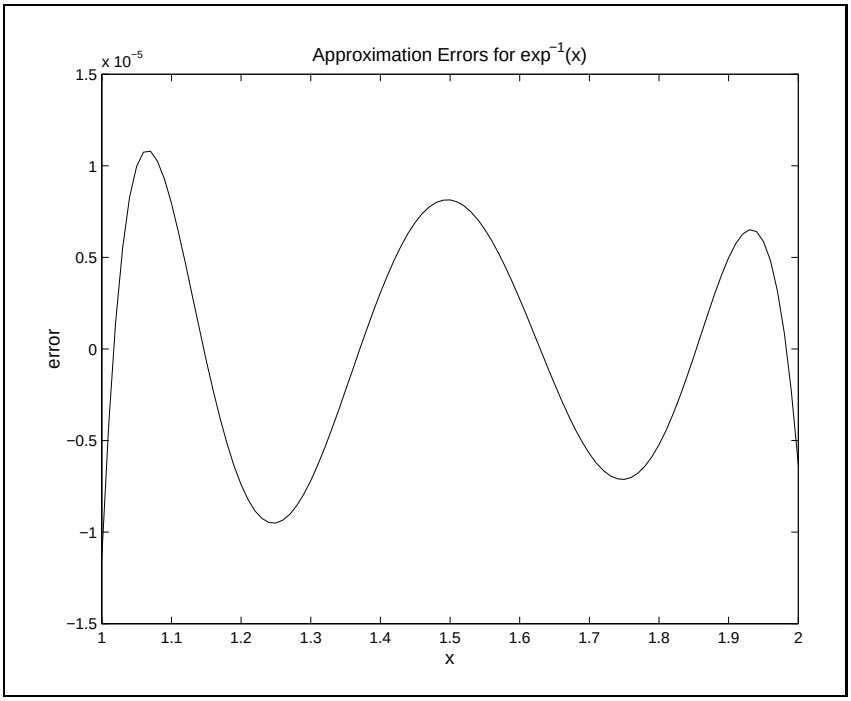


Figure 6.18

and $c$ is an $n \times d$ matrix of coefficients. The collocation strategy selects a set of $n-1$ nodal values of $t$, $t_i$, and finds the value of $c$ that solves the $(n-1)d$ values of the residual function

$$r(t_i, \phi(t_i)c, \phi'(t_i)c) = 0,$$

$i = 1, \ldots, n-1$ and the boundary conditions

$$b_i(t_i^b, \phi(t_i^b)c, \phi'(t_i^b)c) = 0$$

for $i = 1, \ldots, d$. This provides a total of $nd$ equations in $nd$ unknowns.

A general routine for solving first order BVPs is quite simple to design. We will illustrate the use of our solver with a simple example:

$$r(t, x(t), x'(t)) = x'(t) - x(t)A,$$

where

$$A = \begin{bmatrix} -1 & -0.5 \\ 0 & -0.5 \end{bmatrix}$$

with "boundary conditions" $x_1(0) = 1$ and $x_2(1) = 1$. It should be noted that $x$ is defined to be a row vector $(1 \times d)$. We shall seek an approximation on $t \in [0, 2]$; this illustrates the idea that the "boundary" conditions need not be at the boundaries of the domain of interest (they must, however, not be outside of it). The example has a closed form solution $x_1(t) = e^{-t}$ and $x_2(t) = ce^{-t/2} + e^{-t}$, where $c = e^{0.5}(1 - e^{-1})$.

There are two distinct pieces of information that the user must supply. First, the model must be defined. The model consists of the location of the boundary points together with the functions $r(t, x(t), x'(t))$ and $b(t^b, x(t^b), x'(t^b))$. In addition, the model may be defined in terms of a set of parameters used by these functions. To specify a model, the user should define a structure variable with fields `func`, `tb` and `params`. The first field contains the name of a function file that will calculate $r$ and $b$ (described below). The `tb` field is a $d$-vector of points at which $b$ is evaluated. The `params` field should be a cell array of any parameters that are needed to evaluate $r$ and/or $b$; in the example, the cell array will contain the single matrix $A$.

For the example problem, the structure variable is defined by

```
model.func='pbvp01';
model.tb=[0;1];
model.params={A};
```

The function referred to by the `model.func` field computes the residuals and the boundary conditions and should be written using the following syntax:

```
out1=BVPfile(flag,t,x,dx,additional parameters)
switch flag
case 'r'
  out1= residual function evaluated at t, x, dx
case 'b'
  out1= boundary function evaluated at t, x ,dx
end
```

It uses the `flag` variable to determine whether the $r$ or $b$ function is being requested. If the $r$ function is requested, the function is passed an $n-1\times 1$ vector `t` and $n-1\times d$ matrices `x` and `dx`. It should return an $n - 1 \times d$ matrix with $ij$th element equal to $r_j(t_i, x(t_i), x'(t_i))$. If the $b$ function is requested, the function is passed a $d \times 1$ vector `tb` and $d \times d$ matrices `x` and `dx` and should return a $d \times 1$ vector with $i$th element equal to $b_i(t_i^b, x(t_i^b), x'(t_i^b)) = 0$. In our example problem the file looks like

```
function out1=pbvp01(flag,t,x,dx,A);
switch flag
case 'r'
   out1=dx-x*A;
```

```
  case 'b'
    out1=[x(1,1)-1;x(2,2)-1];
  end
```

The solver also needs to know the desired family of approximating functions, `fspace`, (i.e., a family definition structure as defined by `fundef`) the nodal values of $t$ used for collocation, `tnodes`, and an initial guess of the parameter values, `c`.

The general solver routine looks like:

```
function [c,x,r]=bvpsolve(model,fspace,tnode,c,tvals)
% dimension of problem
  d=size(c,2);

% nodal basis matrices
  Phi=funbas(fspace,tnode);
  Phi1=funbas(fspace,tnode,1);

% boundary point basis matrices
  tb=model.tb;
  phi=funbas(fspace,tb);
  phi1=funbas(fspace,tb,1);

% Call rootfinding algorithm
  c=broyden('bvpres',c(:),[],model,fspace,tnode,Phi,Phi1,tb,phi,phi1);
  c=reshape(c,cdef.n,d);

% compute solution and residual functions
  if nargout>1 & ~isempty(tvals)
    x=funeval(c,cdef,tvals);
    dx=funeval(c,cdef,tvals,1);
    r=feval(model.func,'r',tvals,x,dx,model.params{:});
  end
```

In addition to computing the coefficient matrix, $c$, the procedure is implemented to, optionally, take a vector of time values `tvals` and to return the solution and residual functions at those values (`x` and `r`).

The solver instructs the rootfinding algorithm `broyden` to find the roots of the function `BVPRes`, which in turn calls the `model.func` file to compute the residual and boundary functions.

```
function r=bvpres(c,model,fspace,tnode,Phi,Phi1,tb,phi,phi1);
  n=size(Phi,2);
```

```
m=length(tb);
c=reshape(c,n,m);

% Compute residuals at nodal values
x=Phi*c;
dx=Phi1*c;
r=feval(model.func,'r',tnode,x,dx,model.params{:});

% Compute boundary conditions and concatenate to residuals
x=phi*c;
dx=phi1*c;
b=feval(model.func,'b',tb,x,dx,model.params{:});
r=[r(:);b(:)];
```

The demonstration file `dembvp01` contains the code to solve the example problem using Chebyshev polynomial approximants and plots both the approximation error functions and the residual functions. The procedure solves in a single iteration of the rootfinding algorithm because it is a linear problem. An economic application of these procedures is illustrated next with a simple market equilibrium example.

**Example: Commodity Market Equilibrium**

At time $t = 0$ there are available for consumption $S_0$ units of a periodically produced commodity. No more of the good will be produced until time $t = 1$, at which time all of the currently available good must be consumed. The change in the level of the stocks is the negative of the rate of consumption, which is given by the demand function, here assumed to be of the constant elasticity type:

$$s'(t) = -q = -D(p) = -p^{-\eta}.$$

To prevent arbitrage and to induce storage, the price must rise at a rate that covers the cost of capital, $r$ and the physical storage charges, $C$:

$$p'(t) = rp + C.$$

It is assumed that no stocks are carried into the next production cycle, which begins at time $t = 1$; hence the boundary condition that $s(1) = 0$.

This is a two variable system of first order differential equations with two boundary conditions, one at $t = 0$ and the other at $t = 1$. Defining $x = [p; s]$, the residual function is

$$r(t, x, x') = x' - [rx_1 + C \quad - x_1^{-\eta}]$$

and the boundary conditions are $x_2(0) - S_0 = 0$ and $x_2(1) = 0$.

The model structure can be created using

```
model.func='pbvp02';
model.tb=[0;1];
model.params={A};
```

The problem definition file `pbvp02` for this problem is

```
function out1=pbvp02(flag,t,x,dx,r,C,eta,S0);
  switch flag
  case 'r'
    out1=dx-[r*x(:,1)+C  -x(:,1).^(-eta)];
  case 'b'
    out1=x(:,2)-[S0;0];
  end
```

A demonstration file, `dembvp02` is available that uses the parameters $r = 0.10$, $C = 0.5$, $\eta = 2$ and $S_0 = 1$. It approximates the solution using a degree $n = 6$ Chebyshev polynomial approximation. The resulting solution and residual functions are shown in Figures 6.19 and 6.20. It is evident in the latter that the approximation

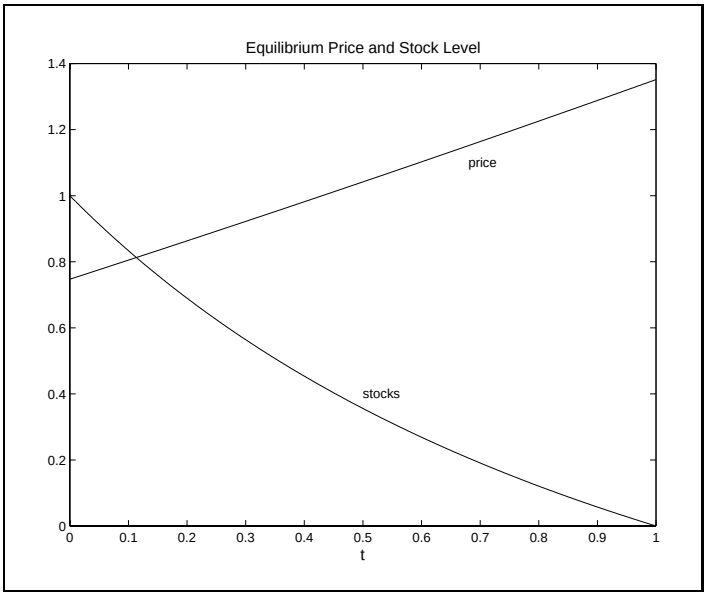


Figure 6.19

achieves a high degree of accuracy even with a low order approximation; the maximum sizes of the price and stocks residual functions are approximately $10^{-10}$ and $10^{-3}$, respectively.

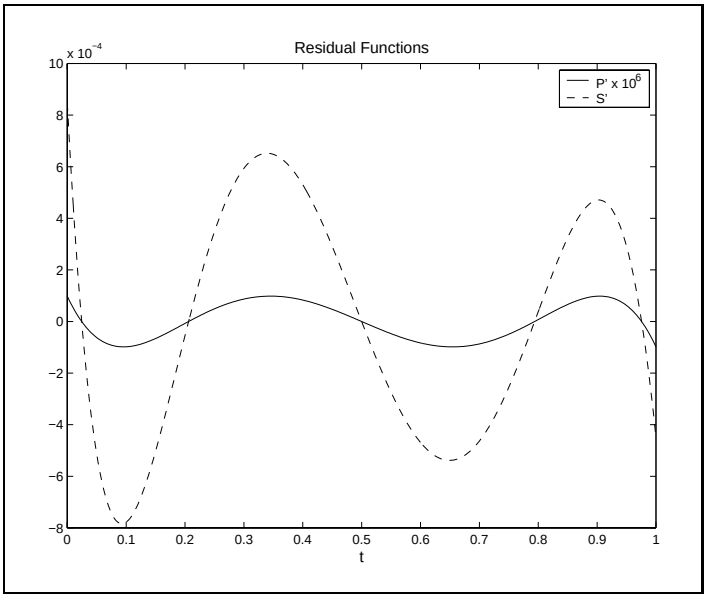
Residual Functions
x 10^-4
P' x 10^6
S'
10
8
6
4
2
0
−2
−4
−6
−8
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
t

Figure 6.20

**Exercises**

6.1. 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 201 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

6.2. 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 its competitiors' output decision functions 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

$$\frac{dp}{dq_i} = \frac{1}{D'(p) - \sum_{j \neq i} S_j'(p)}.$$

In an industry with $m$ identical firms, each firm assumes the other firms will react in the same way it does, so this expression simplifies to

$$\frac{dp}{dq} = \frac{1}{D'(p) - (m-1)S'(p)}.$$

This expression differs from the Cournot case in the extra term in the denominator (which only equals 0 in the monopoly situation of $m = 1$). Notice also that, unlike the Cournot case, the firm's "supply" function depends on the number of firms in the industry.

Write a function to solve this problem analogous to the one described in Section 6.8.1 (page 159) and a demo file to produce the analogous plots. The

function must take the parameters (including $m$, the industry size) and it must also compute the derivative of the $q = S(p)$ function to compute the residual function.

6.3. Consider the potato market model discussed in the Chapter 3 (page 60). 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.

6.4. 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:

$$a = 0.5 + 0.5E[p_1].$$

The rational expectations equilibrium acreage level and expected price satisfy the acreage supply function and

$$E[p_1] = 0.5f(0.75a) + 0.5f(1.25a)$$

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.

6.5. Using collocation with the basis functions of your choice and without using `BVPSOLVE` 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. What degree of approximation is needed to achieve this level of accuracy.

6.6. Lifetime Consumption

A simple model of lifetime savings/consumption choice considers an agent with a projected income flow by $w(t)$, who must choose a consumption rate $c(t)$ to maximize discounted lifetime utility:

$$\max_{C(t)} \int_0^T e^{-\rho t} U(C(t)) dt$$

subject to an intertemporal wealth constraint $dW/dt = rW + w(t) - C$, where $r$ is the rate of return on investments (or the interest rate on borrowed funds, if $W < 0$). The solution to this optimal control problem can be expressed as the system of differential equations

$$C' = -\frac{U'(C)}{U''(C)}(r - \rho)$$

and

$$W' = rW + w(t) - C.$$

It is assumed that the agent begins with no wealth ($W(0) = 0$) and leaves no bequests ($W(T) = 0$).

a) Use `BVPSOLVE` to solve this BVP using the CARA utility function $U(C) = (C^{1-\lambda} - 1)/(1 - \lambda)$ and the parameters values $T = 45$, $r = 0.1$, $\rho = 0.6$, $\lambda = 0.5$ and $w(t) = w_0/(1 + e^{-\alpha t})$, with $w_0 = 1$ and $\alpha = 0.15$. Plot the solution function and the residual functions.

b) In part (a) the agent works until time $T$ and then dies. Suppose, instead, that the agent retires at time $T$ and lives an additional $R = 20$ retirement years with no additional income ($w(t) = 0$ for $T < t \leq T + R$). Resolve the problem with this assumption. What additional problem is encountered? How can the problem be addressed?

6.7. The complementary Normal CDF is defined as

$$\Phi^c(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-z^2/2} dz.$$

Define

$$u(x) = e^{x^2/2} \Phi^c(x).$$

(a) Express $u$ as a differential equation with boundary condition $u(\infty) = 0$.

(b) Use the change of variable $t = x/(K + x)$ (for some constant K) to define a differential equation for the function $v(t) = u(x)$, for $v \in [0, 1]$.

(c) Write a MATLAB function to solve this differential equation using collocation with Chebyshev polynomials. Do this by writing the collocation equations in the form $Bc = f$, where $c$ is an $n$-vector of coefficients. Then solve this linear system directly.

(d) Plot the residual function for a range of values of $K$ between 0.1 and 20. Make a recommendation about the best choice of $K$.

6.8. Write a MATLAB function that automates the approximation of function inverses. The function should have the following syntax:

```
function c=finverse(f,fspace,varargin)
```

You will also need to write an auxiliary function to compute the appropriate residuals used by the rootfinding algorithm.

## Bibliographic Notes

Most introductory texts on numerical analysis contain some discussion of interpolation via Chebyshev polynomials and splines; see, for example, Press et al., or, for a discussion focused on solving differential equations, see Golub and Ortega (Chapter 6).

Collocation is one of a more general class of approximation methods known as weighted residual methods. The general idea of weighted residual methods is to find an approximate that minimizes the residual function for some functional norm. In addition to collocation, two common approaches of this general class are least squares methods, which (for the simple functional equation problem, solve:

$$\min_c \int_a^b r^2(x, \phi(x)c)dx$$

and Galerkin methods (also called Bubnov-Galerkin methods), which solve

$$\int_a^b r(x, \phi(x)c)\phi_i(x)dx = 0, \text{ for } i = 1, \ldots, n.$$

When the integrals in these expressions can be solved explicitly, they seem to be somewhat more efficient than collocation, especially the Galerkin approach. Unless $r$ has a convenient structure, however, these methods will necessitate the use of some kind of discretization to compute the necessary integrals, reducing any potential advantages these methods may have relative to collocation.

# 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 all but a small number of 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 simplest 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 at each point in time 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 expected present 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 $V_{T+1}(s_{T+1})$ in the subsequent period that depends on the realization of the state in that period. 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 $V_{T+1}$, 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, that is, the optimal action may not be unique.

If the decision problem has an infinite horizon, the value functions will not depend on time $t$. We may, therefore, disregard the time subscripts and denote the common value function by $V$. Bellman's equation therefore becomes the vector 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 $\delta$ 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 state space, the action space, the reward function, the state transition function or state transition probabilities, the discount factor $\delta$, 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 of operation. The price of extracted ore is $p$ dollars per ton and the total cost of extracting $x$ tons of ore in any year is $c = x^2/(1+s)$ dollars, where $s$ is the tons of ore remaining in the mine at the beginning of the year. The mine currently contains $\bar{s}$ tons of ore. Assuming the amount of ore extracted in any year must be an integer number of tons, what extraction schedule maximizes profits?

This is a finite horizon, deterministic model with time $t = \{1, 2, \ldots, T\}$ measured in years. The state variable

$$s \in S = \{0, 1, 2, \ldots, \bar{s}\}$$

denotes tons of ore remaining in the mine at the beginning of the year. The action variable

$$x \in X(s) = \{0, 1, 2, \ldots, s\}$$

denotes tons of ore extracted over the year. The state transition function is

$$s' = g(s, x) = s - x.$$

The reward function is

$$f(s, x) = px - x^2/(1 + s).$$

[1] Value functions in infinite horizon problems could be time dependent if $f$, $P$, or $\delta$ 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.

The value of the mine, given it contains $s$ tons of ore at the beginning of year $t$, 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 Asset Replacement - I

At the beginning of each year, a manufacturer must decide whether to continue operating with an aging physical asset or replace it with a new one. An asset that is $a$ years old yields a profit contribution $p(a)$ up to $n$ years, after which the asset becomes unsafe and must be replaced by law. The cost of a new asset is $c$. What replacement policy maximizes profits?

This is an infinite horizon, deterministic model with time $t = \{1, 2, \ldots, T\}$ measured in years. The state variable

$$a \in A = \{1, 2, \ldots, n\}$$

denotes the age of the asset in years. The action variable

$$x \in X(a) = \begin{cases} \{\text{keep}, \text{replace}\} & a < n \\ \{\text{replace}\} & a = n \end{cases}$$

denotes the keep-replacement decision. The state transition function is

$$a' = g(a, x) = \begin{cases} a+1 & x = \text{keep} \\ 1 & x = \text{replace}. \end{cases}$$

The reward function is

$$f(a, x) = \begin{cases} p(a) & x = \text{keep} \\ p(0) - c & x = \text{replace}. \end{cases}$$

The value of an asset of age $a$ satisfies Bellman's equation

$$V(a) = \max\{p(a) + \delta V(a+1), p(0) - c + \delta V(1)\}.$$

Bellman's equation asserts that if the manufacturer keeps an asset of age $a$, he earns $p(a)$ over the coming year and begins the subsequent year with an asset worth $V(a+1)$; if he replaces the asset, on the other hand, he earns $p(0) - c$ over the coming year and begins the subsequent year with an asset worth $V(1)$. Actually, our language is a little loose here. The value $V(a)$ measures not only the current and future net earnings of an asset of age $a$, but also the net earnings of all future assets that replace it.

### 7.2.3 Asset Replacement - II

Consider the preceding example, but suppose that the productivity of the asset may be enhanced by performing annual service maintenance. Specifically, at the beginning of each year, a manufacturer must decide whether to replace the asset with a new one or, if he elects to keep the old one, whether to service it. An asset that is $a$ years old and has been serviced $s$ times yields a profit contribution $p(a, s)$ up to and age of $n$ years, after which the asset becomes unsafe and must be replaced by law. The cost of a new asset is $c$ and the cost of servicing an existing asset is $k$. What replacement policy maximizes profits?

This is an infinite horizon, deterministic model with time $t = \{1, 2, \ldots, T\}$ measured in years. The state variables

$$\begin{aligned} a &\in A = \{1, 2, \ldots, n\} \\ s &\in S = \{0, 1, \ldots, n\} \end{aligned}$$

denote the age of the asset in years and the number of servicings it has undergone, respectively. The action variable

$$x \in X(a, s) = \begin{cases} \{\text{replace, service, no action}\} & a < n \\ \{\text{replace}\} & a = n. \end{cases} ;$$

The state transition function is

$$(a', s') = g(a, s, x) = \begin{cases} (1, 0) & x = \text{replace} \\ (a + 1, s + 1) & x = \text{service} \\ (a + 1, s) & x = \text{no action.} \end{cases}$$

The reward function is

$$f(a, s, x) = \begin{cases} p(0, 0) - c & x = \text{replace} \\ p(a, s + 1) - k & x = \text{service} \\ p(a, s) & x = \text{no action.} \end{cases}$$

The value of asset of age $a$ that has undergone $s$ servicings must satisfy Bellman's equation

$$\begin{aligned} V(a, s) = \max\{ \quad & p(0, 0) - c + \delta V(1, 0), \\ & p(a, s + 1) - k + \delta V(a + 1, s + 1), \\ & p(a, s) + \delta V(a + 1, s)\}. \end{aligned}$$

Bellman's equation asserts that if the manufacturer keeps an asset of age $a$, he earns $p(a)$ over the coming year and begins the subsequent year with an asset worth $V(a+1)$; if he replaces the asset, on the other hand, he earns $p(0) - c$ over the coming year and begins the subsequent year with an asset worth $V(1)$. Actually, our language is a little loose here. The value $V(a)$ measures not only the current and future earnings of an asset of age $a$, but also the optimal earnings of all future assets that replace it.

### 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, $\sigma$ is the annualized volatility of the commodity price, and $\Delta t$ 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]

$$\begin{aligned} p &= \text{commodity price} \\ p &\in S = \{p_1 u^i | i = -N-1, -N, \ldots, N, N+1\}. \end{aligned}$$

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

$$f(p,x) = \begin{cases} 0 & x = \text{keep} \\ \bar{p} - p & x = \text{exercise} \end{cases}$$

[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 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

$V(s)$ = Value of being in employment state $s$ at beginning of week,

must satisfy Bellman's equation

$$V(s) = \begin{cases} \max\{v+\delta V(0), u+\delta pV(1)+\delta(1-p)V(0)\}, & s=0 \\ \max\{v+\delta V(0), w+\delta qV(0)+\delta(1-q)V(1)\}, & s=1 \end{cases}$$

### 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 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 $\gamma$ is the capital survival rate (1 minus the depreciation rate), $f$ is the aggregate production function, and $\epsilon_{t+1}$ 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

$$V(s) = \max_{0 \leq x \leq s} \{u(s-x) + \delta E V(\gamma x + \epsilon f(x))\}, \qquad s > 0.$$

### 7.2.8 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 net social surplus 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

$$V(s) = \max_{0 \le x \le s} \{ \int_0^x p(\xi)\ d\xi - c(x) + \delta V(g(s-x)) \}.$$

### 7.2.9 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 $\delta$, 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 $\delta$, convergence tolerance $\tau$, 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 $\delta$ 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 $\delta$, 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.

The backward recursion, function iteration, and policy iteration algorithms are structured as a series of three nested loops. The outer loop represents either a backward recursion, function iteration, or policy iteration; the middle loop represents visits to each state; and the inner loop represents visits to each action. The computational effort needed to solve a discrete Markov decision model is roughly proportional to the product of the number of times each loop must be executed. More precisely, if $n_s$ is the number of states and $n_x$ is the number of actions, then $n_s \cdot n_x$ total actions need to be evaluated with each outer iteration.

The computational effort needed to solve a discrete Markov decision model is particularly sensitive to the dimensionality of the state and action variables. Suppose, for the sake of argument, that the state variable is $k$-dimensional and each dimension of the state variable has $l$ different levels. Then the number of states will equal $n_s = l^k$. This implies that the computational effort required to solve the discrete Markov decision model will grow exponentially, not linearly, with the dimensionality of the state space. The same will be true regarding the dimensionality of the action space. The tendency for the solution time to grow exponentially with the dimensionality of the state or action space is called the "Curse of Dimensionality". Historically, the curse has represented the most severe practical problem encountered in solving discrete Markov decision models.

## 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 func-

tion $x^*$, 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 $\pi$ 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 A Discrete Dynamic Programming Toolbox

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 five 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, a positive scalar less than one.
- `reward` - An $n$ by $m$ matrix of rewards whose rows and columns are associated with states and actions, respectively.

- `transition` - An $mn$ by $n$ matrix of state transition probabilities whose rows are associated with this period's state and whose columns are associated with 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, if model has a finite horizon, or initial guess for value function, if model has an infinite horizon. It has a default value of zero if not specified.

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 policy `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 MATLABscript 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 MATLABscript 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 MATLABscript 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 MATLAButilities, 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.

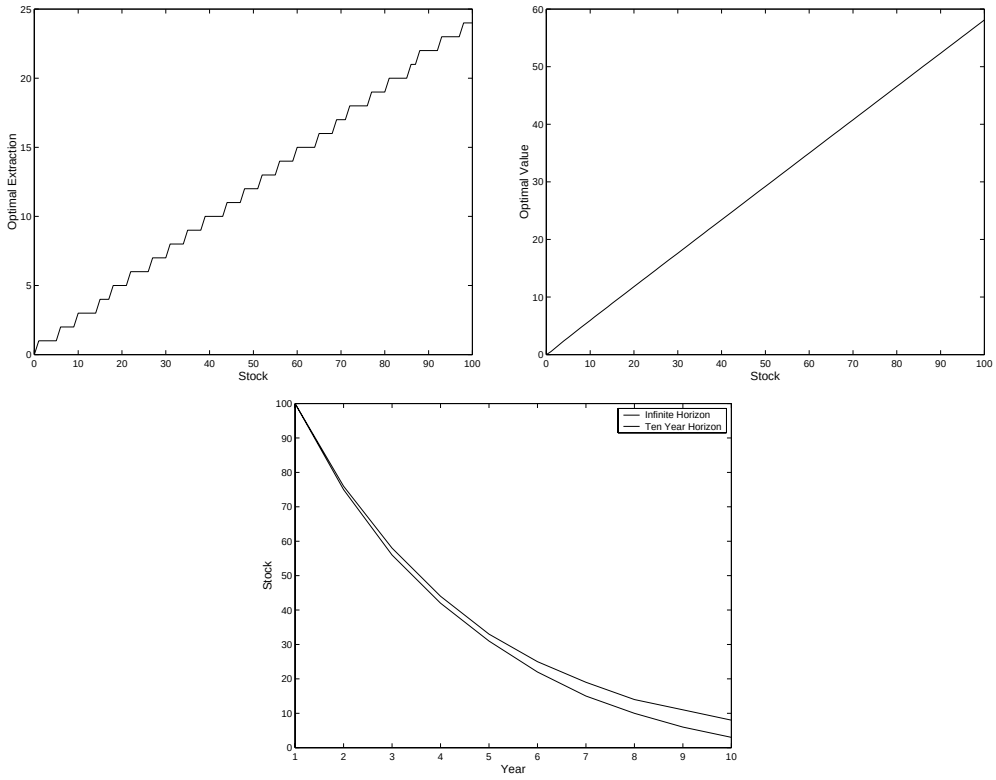


Figure 7.1: Solution to Mine Management Problem

### 7.6.2 Asset Replacement - I

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 |

Then, letting 0=keep and 1=replace, the optimal replacement policy over a five year planning horizon, with no discounting, is:

| | Optimal Policy | | | | | Optimal Value | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Machine Age | | | | | Machine Age | | | | |
| Year | 0 | 1 | 2 | 3 | 4+ | 0 | 1 | 2 | 3 | 4+ |
| 5 | 0 | 0 | 0 | 0 | 0 | 50.0 | 45.0 | 35.0 | 20.0 | 0.0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 95.0 | 80.0 | 55.0 | 20.0 | 0.0 |
| 3 | 0 | 0 | 1 | 1 | 1 | 130.0 | 100.0 | 70.0 | 55.0 | 35.0 |
| 2 | 0 | 1 | 1 | 1 | 1 | 150.0 | 115.0 | 105.0 | 90.0 | 70.0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 165.0 | 150.0 | 125.0 | 110.0 | 90.0 |

Assuming a discount factor of 0.9, the initial year optimal policy and value for functions for differing horizons are:

| | Optimal Policy | | | | | Optimal Value | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Machine Age | | | | | Machine Age | | | | |
| Horizon | 0 | 1 | 2 | 3 | 4+ | 0 | 1 | 2 | 3 | 4+ |
| 1 | 0 | 0 | 0 | 0 | 0 | 50.0 | 45.0 | 35.0 | 20.0 | 0.0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 90.5 | 76.5 | 53.0 | 20.0 | 0.0 |
| 3 | 0 | 0 | 1 | 1 | 1 | 118.8 | 92.7 | 56.4 | 41.4 | 21.4 |
| 4 | 0 | 0 | 1 | 1 | 1 | 133.4 | 95.8 | 82.0 | 67.0 | 47.0 |
| 5 | 0 | 0 | 0 | 1 | 1 | 136.2 | 118.8 | 95.3 | 80.1 | 60.1 |
| 6 | 0 | 0 | 0 | 1 | 1 | 156.9 | 130.7 | 107.1 | 82.6 | 62.6 |
| 7 | 0 | 0 | 1 | 1 | 1 | 167.7 | 141.4 | 116.2 | 101.2 | 81.2 |
| 8 | 0 | 0 | 0 | 1 | 1 | 177.2 | 149.6 | 126.1 | 110.9 | 90.9 |
| 9 | 0 | 0 | 0 | 1 | 1 | 184.6 | 158.5 | 134.8 | 119.5 | 99.5 |
| 10 | 0 | 0 | 0 | 1 | 1 | 192.6 | 166.3 | 142.6 | 126.2 | 106.2 |
| 20 | 0 | 0 | 0 | 1 | 1 | 237.2 | 210.3 | 186.5 | 171.0 | 151.0 |
| 40 | 0 | 0 | 0 | 1 | 1 | 257.9 | 231.3 | 207.4 | 191.8 | 171.8 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 60 | 0 | 0 | 0 | 1 | 1 | 260.5 | 233.9 | 209.9 | 194.4 | 174.4 |
| 100 | 0 | 0 | 0 | 1 | 1 | 260.8 | 234.2 | 210.2 | 194.7 | 174.7 |
| 200 | 0 | 0 | 0 | 1 | 1 | 260.8 | 234.2 | 210.2 | 194.7 | 174.7 |

The optimal steady-state policy is to replace the tractor after year three.

### 7.6.3 Asset Replacement - II

Consider the same model as above, except that now the profit contribution of a tractor

$$f(a, n) = (50.0 - 2.5a - 2.5a^2) * (1 - (a - n)/4)$$

depends both on its age a and the number of times n it has undergone end-of-year servicing. At the beginning of the year, a farmer must decide what to do at the end of the year: keep and service the tractor, keep but not service the tractor, or replace the tractor. It costs \$75 to order a new tractor and \$10 to schedule one for servicing. Assuming a discount factor of 0.9, the steady-state optimal replacement-maintenance policy and value functions are:

| | Optimal Policy | | | | | Optimal Value | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Times Serviced | | | | | Times Serviced | | | | |
| Age | 0 | 1 | 2 | 3 | 4 | 0 | 1 | 2 | 3 | 4 |
| 0 | 2 | 0 | 0 | 0 | 0 | 163.2 | 0.0 | 0.0 | 0.0 | 0.0 |
| 1 | 1 | 2 | 0 | 0 | 0 | 114.2 | 136.8 | 0.0 | 0.0 | 0.0 |
| 2 | 3 | 1 | 1 | 0 | 0 | 89.3 | 99.9 | 113.2 | 0.0 | 0.0 |
| 3 | 3 | 3 | 3 | 3 | 0 | 76.8 | 81.8 | 86.8 | 91.8 | 0.0 |
| 4 | 3 | 3 | 3 | 3 | 3 | 71.8 | 71.8 | 71.8 | 71.8 | 71.8 |

where 0=not defined, 1=keep but don't service, 2=keep and service, and 3=replace.

The optimal steady-state policy is thus to service the tractor after its first and second years, to keep it but not service it after its third year, and to replace it after its fourth year. Should one forget to service the tractor after its first year, then it would be optimal to keep but not service it after its second year and then to sell it after its third year.

### 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 $N = 50$ 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 7.2.

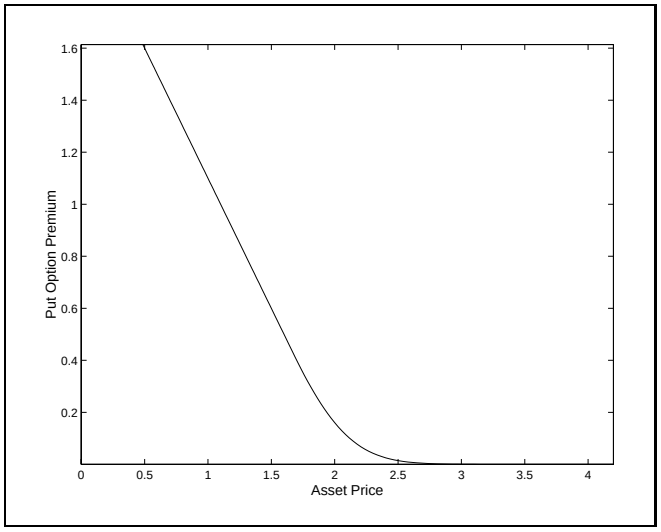


Figure 7.2

### 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 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 7.1.

Table 7.1: 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 7.3.

### 7.6.7 Optimal Growth

### 7.6.8 Renewable Resource Problem

### 7.6.9 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:

```
T = 10;                         % foraging periods
eadd =  4;                      % energy from foraging
```

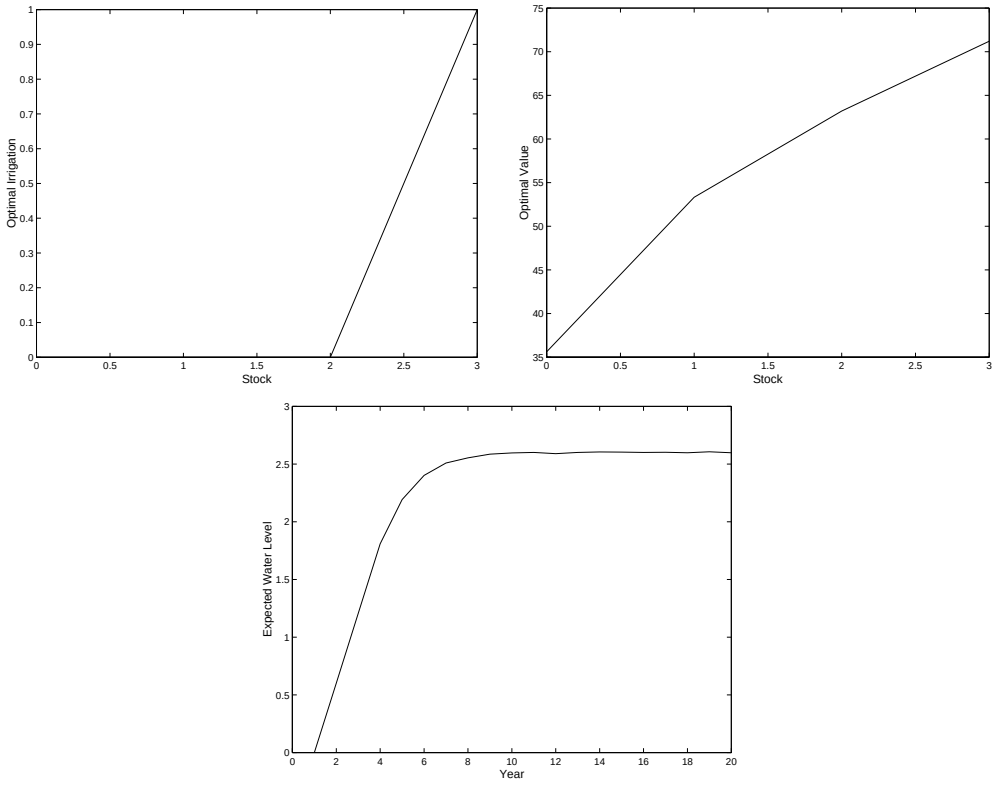


Figure 7.3: Solution to Optimal Irrigation Problem

```
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 7.2):

```
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 7.3).

Table 7.2: 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 |

Table 7.3: 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 |

**Exercises**

7.1. 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.

7.2. A machine lasts a maximum of 6 years but may require replacement sooner. Suppose that the probability of requiring replacement after each year 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 is the age distribution of macines in a large population? Draw a histogram.

(b) What is the average machine age in a large population?

7.3. 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.

7.4. Consider a competitive price-taking firm that wishes to maximize the present value sum of current and future profits from harvesting a nonrenewable 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 $\alpha x_t^{\beta}$, where $\alpha$ and $\beta$ 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:

$$\Pr[p_{t+1} = p_j | p_t = p_i] = w_{ij}.$$

Assuming an annual discount factor of $\delta$, 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.

7.5. 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 $\delta$.

(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.

7.6. A firm operates in an uncertain profit environment. At the beginning of each period $t$, the firm observes its potential short-run variable profit $\pi_t$, which may be negative, and then decides whether to operate, making a short-run variable profit $\pi_t$, 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 $\pi_t$ 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?

7.7. 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?

7.8. 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 MATLABprogram 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.

7.9. 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:

$$\begin{array}{rcl} \Pr[p_{t+1} = 1 | p_t = 1] & = & 0.8 \\ \Pr[p_{t+1} = 2 | p_t = 1] & = & 0.2 \\ \Pr[p_{t+1} = 1 | p_t = 2] & = & 0.3 \\ \Pr[p_{t+1} = 2 | p_t = 2] & = & 0.7 \end{array}$$

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 MATLABprogram 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?

7.10. 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.

7.11. 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.

7.12. 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?

7.13. At dairy producer must decide whether to keep and lactate a cow or replace it with a new one. A cow yields $y_i = 8 + 2i - 0.25i^2$ tons of milk over its $i^{th}$ lactation up to ten lactations, after which she becomes barren and must be replaced. Assume that the net cost of replacing a cow is 500 dollars, the profit contribution of milk is 150 dollars per ton, and the per-laction discount factor is $\delta = 0.9$.

(a) What lactation-replacement policy maximizes profits?

(b) What is the optimal policy if the profit contribution of milk rises to 200 dollars per ton?

(c) What is the optimal policy if the cost of replacement $c_t$ follows a three-state Markov chain with possible values, \$400, \$500, and \$600, and transition probabilities

| | $c_{t+1}$ | | |
|---|---|---|---|
| $c_t$ | \$400 | \$500 | \$600 |
| \$400 | 0.5 | 0.4 | 0.1 |
| \$500 | 0.2 | 0.6 | 0.2 |
| \$600 | 0.1 | 0.4 | 0.5 |

# Chapter 8

# Discrete Time Continuous State Dynamic Models: Theory

We now turn our attention to discrete time dynamic economic models whose state variables may assume a continuum of values. Three classes of discrete time continuous state dynamic economic models are examined. One class includes models of centralized decision making by individuals, firms, or institutions. Examples of continuous state dynamic decision models involving discrete 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. Examples of continuous state decision models admitting a continuum of choices include a central planner managing the harvest of a natural resource, an entrepreneur planning production and investment, and a consumer making consumption and savings decisions.

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 affects the immediate and long-run welfare of another. Examples of such models include national grain marketing boards deciding how much grain to sell on world markets, producers of substitute goods deciding whether to expand factory capacity, and individuals deciding how much work effort to exert within 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 hap-

pen at a future date. If it is assumed that agents' expectations are consistent with the implications of the model, then agents are said to possess rational expectations. Rational expectations models may be used to study asset returns in a pure exchange economy, futures prices in a primary commodity market, and agricultural producer responses to government price support programs.

Dynamic optimization and equilibrium models are closely related. The solutions to continuous state continuous action dynamic optimization models may often be equivalently characterized by first-order intertemporal equilibrium conditions obtained by differentiating the 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 introduces the theory of discrete time continuous state dynamic economic models and provides illustrative examples. The subsequent chapter is devoted to numerical methods that may be used to solve and analyze such models.

## 8.1 Continuous State Dynamic Programming

The discrete time continuous state Markov decision model has the following structure: In every period $t$, an agent observes the state of an economic process $s_t \in S$, takes an action $x_t \in X$, and earns a reward $f(s_t, x_t)$ that depends on both the state of the process and the action taken. 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 $\epsilon_{t+1}$ that is unknown in period $t$:

$$s_{t+1} = g(s_t, x_t, \epsilon_{t+1}).$$

The agent seeks a policy of state-contingent actions $x_t^* : S \mapsto X$, $t = 0, 1, 2, \ldots, T \leq \infty$, that maximizes the present value of current and expected future rewards, discounted at a per-period factor $\delta$:

$$\max_{\{x_t(\cdot)\}} E_0 \sum_{t=0}^{T} \delta^t f(s_t, x_t(s_t)).$$

The state vector $s \in \Re^n$ of a continuous state Markov decision model may possess continuous state variables whose ranges are intervals of the real line and discrete state variables whose ranges are finite subsets of real numbers. At least one state variable must be continuous. If all states are continuous, the state space is said to be purely continuous; if some states are continuous and some are discrete, the state space is

said to be mixed. The state space, which contains all possible realizations of the state vector, is denoted $S \subseteq \Re^n$.

The action vector $x \in \Re^m$ of a continuous state Markov decision model may possess continuous action variables whose ranges are intervals of the real line or discrete action variables whose ranges are finite subsets of real numbers. We limit our discussion, however, to models whose actions are all of one form or the other. If all actions are continuous, the model is said to be a continuous choice model; if all actions discrete, the model is said to be a discrete choice model. The action space, which contains all possible realizations of the action vector, is denoted $X \subseteq \Re^m$.

In the continuous state Markov decision model, the exogenous random shocks $\epsilon_t$ are assumed to be independently and identically distributed over time, and independent of preceding states and actions. The reward functions $f$ and the state transition functions $g$ are assumed to be twice continuously differentiable over the continous dimensions of $S$ and $X$ and the per-period discount factor $\delta$ 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 $X(s)$.

Like the discrete Markov decision problem, the discrete time continuous state 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 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, $V_t(s)$ denotes the maximum attainable sum of current and expected future rewards, given that the process is in state $s$ in period $t$. The functions $V_t : S \mapsto \Re$ are called the value functions. The Bellman equation succinctly captures the essential problem faced by a dynamically optimizing agent: the need to optimally balance immediate benefits $f$ against expected future benefits $\delta E V_{t+1}$.

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 Markov decision model may be solved recursively, 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 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 $\delta$ 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.

## 8.2 Continuous State Discrete Choice Models

Discrete choice dynamic decision models are common in management, economics, and finance. Many of these models involve binary choices, in which an agent must decide whether or not to undertake a specific action. Regenerative binary choice models involve decisions that bring an economic process back to some natural initial state, from which the decision cycle begins anew. Examples of such models include asset replacement, in which the agent starts fresh with a new asset whenever an old one is replaced, and timber cutting, in which the agent starts fresh with a stand of seedlings after cutting down the old stand. Non-regenerative binary decision models involve decisions that bring an economic process to some natural terminal state, from which the process never again emerges and further decisions are moot. An example of such a model is the put option, which, once exercised, can never be exercised again.

### 8.2.1 Asset Replacement

A producer must decide when to replace an aging asset with a new one. The asset produces $q(a)$ units of output in its $a^{th}$ period of operation, up to period $\bar{a}$, after which its output drops to zero. A new asset costs $K$ and the profit contribution per unit of output $p$ is an exogenous continuous-valued Markov process

$$p_{t+1} = g(p_t, \epsilon_{t+1}).$$

What is the optimal machine replacement policy and what is the value of owning an asset of age $a$ if the price of output is $p$?

This is an infinite horizon, stochastic model with one continuous state variable, the current unit profit contribution $p \in (0, \infty)$, and one discrete state variable, the current age of the asset $a \in \{1, 2, \ldots, \bar{a}\}$. The model has a single binary choice

variable $j$, which equals 1 if the asset is replaced and equals 0 otherwise. The current reward earned by the producer equals the profit contribution generated by the asset less replacement costs, if any:

$$f(s,a,j) = \begin{cases} p\,q(0) - K & j = 1 \\ p\,q(a) & j = 0. \end{cases}$$

The value $V(p,a)$ of owning an asset of age $a$, given the current unit profit contribution of output is $p$, satisfies Bellman's equation

$$V(p,a) = \max\{p\,q(a) + \delta E_\epsilon V(g(p,\epsilon), a+1), p\,q(0) - K + \delta E_\epsilon V(g(p,\epsilon), 1)\}.$$

### 8.2.2 Timber Cutting

The owner of a timber stand must decide each year whether to cut down the stand and sell it for lumber, or allow the stand to develop one more year. The stand biomass $s$ grows at a deterministic rate:

$$s_{t+1} = g(s_t).$$

The profit contribution per unit of biomass is $p$, treated as constant. Immediately upon cutting down the timber stand, the owner is required to plant seedlings, incurring a replanting cost $K$. What is the optimal timber cutting policy and what is the value of owning a timber stand with biomass $s$?

This is an infinite horizon, stochastic model with one continuous state variables, the current biomass of the timber stand $s \in [0, \infty)$. The model has one binary choice variable $j$, which equals 1 if the stand is cut and replanted and equals 0 otherwise. The current reward earned by the owner equals the revenue from lumber sales less replanting costs, if the stand is cut, but equals zero otherwise:

$$f(s,j) = \begin{cases} p\,s - K & j = 1 \\ 0 & j = 0. \end{cases}$$

The value $V(s)$ of a timber stand of biomass $s$ satisfies Bellman's equation

$$V(s) = \max\{\delta E_\epsilon[V(g(s))], p\,s - K + \delta E_\epsilon[V(0)]\}.$$

### 8.2.3 American 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 $K$ on or before a specified

expiration period $T$. In the discrete-time Black-Scholes option pricing model, the price of the commodity follows an exogenous continuous-valued Markov process

$$p_{t+1} = g(p_t, \epsilon_{t+1}).$$

What is the value of an American put option in period $t$ if the commodity price is $p$? At what critical price is it optimal to exercise the put option and how does this critical price vary over time?

This is a finite horizon, stochastic model with one continuous state variable, the current price of the commodity $p$, and one binary state variable, $i$, which equals 0 if the option has already been exercised or 1 otherwise. The model has one binary choice variable, $j$, which equals 1 if the option is exercised in the current period and 0 otherwise. The current period reward earned by the holder of the option is

$$f(p, i, j) = ij(K - p).$$

The value $V_t(p, 1)$ of an unexercised put option in period $t$, given the commodity price $p$, satisfies Bellman's equation

$$V_t(p, 1) = \max\{K - p, \delta E_\epsilon V_{t+1}(g(p, \epsilon), 1)\},$$

subject to the terminal condition $V_{T+1}(p, 1) = 0$. The value of a previously exercised put option is zero, regardless of the price of the commodity.

### 8.2.4 Industry Entry and Exit

A firm operates in an uncertain profit environment. At the beginning of each period, the firm observes its potential short-run profit over the coming period $\pi$, which may be negative, and decides whether to operate, taking the short run profit $\pi$, or to not operate, making a short-run profit of 0. Although the firm faces no fixed costs, it incurs a shut down cost $K_0$ if it closes after a period of activity, and a start-up cost $K_1$ if it opens after a period of inactivity. The short-run profit $\pi$ is an exogenous continuous-valued Markov process

$$\pi_{t+1} = g(\pi_t, \epsilon_{t+1}).$$

What is the value of the firm and what is the optimal entry-exit policy? In particular, how low must the short-run profit be for active firm to close and how high must the short-run profit be for an inactive firm to open?

This is an infinite horizon, stochastic model with one continuous state variable, the current short-run profit $\pi \in (-\infty, \infty)$, and one binary state variable $i$, which equals 1 if the firm operated in the preceding period or 0 otherwise. The model has a single binary choice variable $j$, which equals 1 if the firm operates in the current

period or 0 otherwise. The current reward earned by the firm is its short-run profit less transactions costs, if any:

$$f(\pi, i, j) = \pi j - K_0 i(1-j) - K_1 j(1-i).$$

The value $V(\pi, i)$ of the firm, given the current short-run profit $\pi$ and the firm's previous operational state $i$, satisfies Bellman's equation

$$V(\pi, i) = \max_{j=0,1}\{\pi j - K_0 i(1-j) - K_1 j(1-i) + \delta E_\epsilon V(g(\pi, \epsilon), j)\}.$$

### 8.2.5 Job Search

An infinitely-lived laborer must make employment decisions in an environment with fluctuating wages and uncertain job security. At the beginning of each period, the laborer observes the current wage rate $w$. If the laborer is currently employed, he must decide whether to continue to work at the given wage or to quit. If the laborer is currently unemployed, he must decide whether to search for job to begin the following period. If the laborer works, he receives the going wage rate $w$; if he searches for a job, he receives an unemployment benefit $u$; and if he neither works nor searches, he receives a benefit from leisure $v$. If the agent searches for a job, his probability of finding employment for the following period is $\pi_f$; if he elects to work, he faces a probability $\pi_k$ of keeping his job for the following period. The wage rate $w$ is an exogenous continuous-valued Markov process

$$w_{t+1} = g(w_t, \epsilon_{t+1}).$$

What is the laborers optimal search and employment policy? In particular, how low must the wage rate be for him to decline to search if unemployed; and how low must the wage rate be for him to quit his job if employed?

This is an infinite horizon, stochastic model with one continuous state variable, the current wage rate $p \in [0, \infty)$, and one binary state variable $i$, which equals 1 if the laborer is employed at the beginning of the period or 0 otherwise. The model has a single binary choice variable $j$, which equals 1 if the laborer is 'active', that is, if he continues to work if employed or initiates a search if unemployed, and which equals 0 otherwise. The current reward earned by the laborer is:

$$f(w, i, j) = wij + u(1-i)j + v(1-j).$$

The value $V(\pi, i)$ of the firm, given the current short-run profit $\pi$ and the firm's previous operational state $i$, satisfies Bellman's equation

$$V(w, i) = \max_{j=0,1}\{f(w, i, j) + \delta E_\epsilon[\pi_{ij} V(g(w, \epsilon), 1) + (1 - \pi_{ij}) V(g(w, \epsilon), 0)]\}$$

where $\pi_{01} = \pi_f$, $\pi_{11} = \pi_k$, and $\pi_{00} = \pi_{10} = 0$.

## 8.3 Continuous State Continuous Choice Models

Markov decision models with continuous state and action spaces are special because their solutions can often be characterized by "first-order" equilibrium conditions. Characterizing the solution to a Markov decision problem via its equilibrium conditions, the so-called Euler conditions, provides an intertemporal arbitrage interpretation that helps the analyst understand and explain the essential features of the optimized dynamic economic process. Below, we derive the Euler conditions for the infinite horizon model, leaving derivation of the Euler conditions for finite horizon models as an exercise.

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 $\lambda$ the shadow price function. It represents the marginal value of the state variable to the optimizer or, equivalently, the price that the optimizer imputes to the state variable.

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 optimization problem imply that the optimal action $x$, given state $s$, 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:

$$f_s(s,x) + \delta E_\epsilon \left[\lambda(g(s,x,\epsilon))g_s(s,x,\epsilon)\right] = \lambda(s).$$

Here, $f_x$, $f_s$, $g_x$, and $g_s$ denote partial derivatives whose dimensions are 1xm, 1xn, nxm, and nxn, respectively, where $n$ and $m$ are the dimensions of the state and action spaces, respectively.

In certain applications, the state transition depends only on the action taken by the agent, 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 eliminates the shadow price function as an unknown, and simplifies the Euler conditions into a single functional equation in a single unknown, the optimal policy function $x$:

$$f_x(s,x(s)) + \delta E_\epsilon \left[f_s(g(s,x(s),\epsilon),x(g(s,x(s),\epsilon)))g_x(s,x(s),\epsilon)\right] = 0.$$

This equation, when it exists, is called the Euler equation.

The Euler conditions take a different form when actions are subject to constraints. Suppose, for example, that actions are subject to bounds of the form

$$a(s) \leq x \leq b(s),$$

where $a : S \mapsto X$ and $b : S \mapsto X$ are differentiable functions of the state $s$. In these instances, the Euler conditions take the form:

$$f_x(s,x) + \delta E_\epsilon \left[\lambda(g(s,x,\epsilon))g_x(s,x,\epsilon)\right] = \mu$$

$$f_s(s,x) + \delta E_\epsilon \left[\lambda(g(s,x,\epsilon))g_s(s,x,\epsilon)\right] + \min(\mu,0)a'(s) + \max(\mu,0)b'(s) = \lambda(s)$$

where $x$ and $\mu$ satisfy the complementarity condition

$$a(s) \leq x \leq b(s), \qquad x_i > a_i(s) \Longrightarrow \mu_i \geq 0, \qquad x_i < b_i(s) \Longrightarrow \mu_i \leq 0.$$

Here, $\mu$ is a 1xm vector whose $i^{th}$ element, $\mu_i$, measures the current and expected future reward from a marginal increase in the $i^{th}$ action variable $x_i$. At the optimum, $\mu_i$ must be nonpositive if $x_i$ is less than its upper bound, for otherwise rewards can be increased by raising $x_i$; similarly, $\mu_i$ must be nonnegative if $x_i$ is greater than its lower bound, for otherwise rewards can be increased by lowering $x_i$. And if $x_i$ is neither at its upper or lower bound, $\mu_i$ must be zero to preclude the possibility of increasing rewards via marginal changes in $x_i$ in either direction.

An analyst is often interested with the long-run 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 $\lambda^*$ that satisfy the Euler and state stationarity conditions:

$$f_x(s^*,x^*) + \delta\lambda^* g_x(s^*,x^*) = 0$$

$$\lambda^* = f_s(s^*,x^*) + \delta\lambda^* g_s(s^*,x^*)$$

$$s^* = g(s^*,x^*).$$

The steady-state conditions of a constrained deterministic dynamic optimization problem can be similarly stated, except that they take the form of a nonlinear complementarity problem, rather than a system of nonlinear equations. Whether the action is constrained or not, the steady-state conditions pose a finite-dimensional problem that can typically be solved numerically using standard nonlinear equation or complementarity methods. In simpler applications, the conditions can often be solved

analytically, even if the Bellman and Euler equations do not possess closed-form solutions. In such situations, it is often further possible through implicit differentiation to derive explicit closed-form expressions for the derivatives of the steady-state state, action, and shadow price with respect to critical model parameters.

Knowledge of the steady-state of a deterministic Markov decision problem is often very useful in applied work. 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 long-run behavior of the process. The analyst 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. For stochastic models, the state and action generally will not converge to specific values and the long-run 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 assist the analyst by providing a reasonable initial guess for the optimal policy, value, and shadow price functions in iterative numerical solution algorithms designed to solve the Bellman equation or Euler conditions. 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.

### 8.3.1 Optimal Economic Growth

Consider an economy that produces and consumes a single composite good. Each year begins with a predetermined amount of the good $s$ in stock, of which an amount $x$ is invested and the remainder $s - x$ is consumed, yielding a social benefit $u(s - x)$. The amount of good available at the beginning of each year is a controlled Markov process

$$s_{t+1} = \gamma x_t + \epsilon_{t+1} f(x_t)$$

where $\gamma$ is the capital survival rate (1 minus the depreciation rate), $f$ is the aggregate production function, and $\epsilon$ is a positive production shock with mean 1. What consumption-investment policy maximizes the sum of current and expected future social benefits?

This is an infinite horizon, stochastic model with one state variable, the stock of good at beginning of the year $s$, and one choice variable, the amount of good invested over the current year $x$, which is subject to the constraint $0 \leq x \leq s$. The sum of

current and expected future social benefits $V(s)$, satisfies Bellman's equation

$$V(s) = \max_{0 \leq x \leq s} \{u(s-x) + \delta E V(\gamma x + \epsilon f(x))\}.$$

Assuming $u'(0) = -\infty$ and $f(0) = 0$, the constraints will never be binding at an optimum and the shadow price of the composite good $\lambda(s)$ will satisfy the Euler equilibrium conditions:

$$u'(s-x) - \delta E\left[\lambda(\gamma x + \epsilon f(x))(\gamma + \epsilon f'(x))\right] = 0$$

$$\lambda(s) = u'(s-x).$$

These conditions imply that along the optimal path

$$u'_t = \delta E_t\left[u'_{t+1}(\gamma + \epsilon_{t+1} f'_t)\right]$$

where $u'_t$ is current marginal utility and $\epsilon_{t+1} f'_t$ is the following period's marginal product of capital. Thus, the utility derived from a unit of good today must equal the discounted expected utility derived from investing it and consuming its yield tomorrow.

The certainty-equivalent steady-state, which is obtained by fixing the production shock $\epsilon$ at its mean 1, are the stock level $s^*$, investment level $x^*$, and shadow price $\lambda^*$ that solve 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: $1 - \gamma + r = f'(x^*)$, where $\delta = 1/(1+r)$. That is, the marginal product of capital must equal the capital depreciation rate plus the interest rate. Totally differentiating the equation system above with respect to the interest rate $r$:

$$\frac{\partial s^*}{\partial r} = \frac{1+r}{f''} < 0$$

$$\frac{\partial x^*}{\partial r} = \frac{1}{f''} < 0$$

$$\frac{\partial \lambda^*}{\partial r} = \frac{u'' r}{f''} > 0.$$

Thus, a permanent rise in the interest rate will reduce the deterministic steady-state stock and investment levels, and will raise the shadow price.

### 8.3.2 Public Renewable Resource Management

A social planner wishes to maximize social benefits derived from harvesting a publicly-owned resource. Each year begins with a predetermined stock of the resource $s$, of which an amount $x$ is harvested at a total cost $c(x)$ and sold at a market clearing price $p(x)$. The remainder $s - x$ is retained for reproduction. The stock of resource available at the beginning of each period follows a controlled deterministic process

$$s_{t+1} = g(s_t - x_t).$$

What harvest policy maximizes the sum of current and future net social surplus? What is the steady-state resource stock and harvest and how do they vary with the interest rate?

This is an infinite horizon, deterministic model with one state variable, the stock of resource at beginning of the period $s$, and one choice variable, the amount of resource harvested over the current period $x$, which is subject to the constraint $0 \leq x \leq s$. The current reward, net social surplus, is derived by integrating under the demand curve and subtracting total harvest costs:

$$f(x) = \int_0^x p(\xi)\ d\xi - c(x).$$

The value $V(s)$ of the 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 constraint will never be binding at an optimum and the shadow price of the resource $\lambda(s)$ will satisfy the Euler equilibrium conditions:

$$p(x) = c'(x) + \delta\lambda(g(s-x))g'(s-x)$$

$$\lambda(s) = \delta\lambda(g(s-x))g'(s-x).$$

These conditions imply that 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 shadow price of the unharvested resource and the marginal cost of harvesting

it. Moreover, the current value of one unit of the resource equals the discounted value of its yield in the subsequent period.

The steady-state resource stock $s^*$, harvest $x^*$, and shadow price $\lambda^*$ 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 rate of growth of resource stock equals the interest rate. Totally differentiating this equation and making reasonable assumptions about the curvature of the growth function $g$:

$$\frac{\partial s^*}{\partial r} = \frac{1+r}{g''} < 0$$

$$\frac{\partial x^*}{\partial r} = \frac{r}{g''} < 0.$$

That is, as the interest rate rises, the steady-state stock and harvest fall.

### 8.3.3 Private Nonrenewable Resource Management

A mine owner wishes to maximize profits derived from extracting and selling ore. Each period begins with a predetermined stock of ore $s$, of which an amount $x$ is extracted at a total cost $c(x)$ and sold at a market price $p$, which is assumed constant. What extraction policy maximizes the sum of current and future profits?

This is an infinite horizon, deterministic model with one state variable, the stock of ore at beginning of the period $s$, and one choice variable, the amount of ore extracted over the current period $x$, which is subject to the constraint $0 \leq x \leq s$. The current reward, net profit, equals revenue from ore sales less the cost of extraction:

$$f(s, x) = p\,x - c(x).$$

The value $V(s)$ of the mine containing ore stock $s$ satisfies Bellman's equation

$$V(s) = \max_{0 \leq x \leq s} \{p\,x - c(x) + \delta V(s - x)\}.$$

If $p > c'(0)$, one can show that ore will always be extracted provided there are stocks remaining. However, it is not possible to rule out the possibility that in

some states it will be optimal to extract all that remains in the mine. That is, the upper bound on $x$ may be binding. As such, the Euler conditions take the form of a complementarity condition. More specifically, the shadow price of the resource $\lambda(s)$ is characterized by the following:

$$p - c'(x) - \delta\lambda(s - x) = \mu$$

$$\lambda(s) = \delta\lambda(s - x) + \max(\mu, 0)$$

where the ore extracted $x$ and the long-run marginal profit of extraction $\mu$ must satisfy the complementarity condition

$$0 \leq x \leq s, \qquad x > 0 \Longrightarrow \mu \geq 0, \qquad x < s \Longrightarrow \mu \leq 0.$$

Thus, in any period, ore is extracted until the long-run marginal profit is driven to zero or the content of the mine is exhausted, whichever comes first.

Under the assumption $p > c'(0)$, the model admits only one steady state: $s^* = x^* = 0$, $\lambda^* = p - c'(0)$, and $\mu^* = (1 - \delta)\lambda^*$. That is, the mine will be worked until its contents are depleted. Until such time that the content of the mine is depleted,

$$p_t = c'_t + \delta\lambda_t$$

$$\lambda_t = \delta\lambda_{t+1}.$$

where $p_t$ is the market price and $c'_t$ is the marginal cost of extraction. That is, the market price of extracted ore equals the shadow price of unextracted ore plus the marginal cost of extraction. Also, the current-valued shadow price of unextracted ore will grow at the rate of interest, or equivalently, the present-value shadow price will remain constant.

### 8.3.4 Optimal Water Management

A water planner wishes to maximize social benefits derived from the water collected in a reservoir. The water may be used either for irrigation or recreation. Irrigation during the spring benefits agricultural producers, but reduces the reservoir level during the summer, damaging recreational users. Specifically, if $s$ is the water level at the beginning of spring and an amount $x$ is released for irrigation, producer benefits will be $a(x)$ and recreational user benefits will be $u(s - x)$. Water levels are replenished during the winter months by i.i.d. random rainfalls $\epsilon$, implying that the reservoir level at the beginning of each year is a controlled Markov process

$$s_{t+1} = s_t - x_t + \epsilon_{t+1}.$$

What irrigation policy maximizes the sum of current and expected future benefits to producers and users combined?

This is an infinite horizon, stochastic model with one state variable, the reservoir water level at beginning of the year $s$, and one choice variable, the amount of water released for irrigation $x$, which is subject to the constraint $0 \leq x \leq s$. The current reward is the sum of producer and user benefits

$$f(s,x) = a(x) + u(s-x).$$

The value of the water in the reservoir satisfies Bellman's equation

$$V(s) = \max_{0 \leq x \leq s} \{a(x) + u(s-x) + \delta E V(s - x + \epsilon)\}.$$

Assuming $a'(0)$ and $u'(0)$ are sufficiently large, the constraints will not be binding at an optimal solution and the shadow price of water $\lambda(s)$ will satisfy the Euler equilibrium conditions

$$a'(x) - u'(s-x) - \delta E \lambda(s - x + \epsilon) = 0$$

$$\lambda(s) = u'(s-x) + \delta E \lambda(s - x + \epsilon).$$

It follows that along the optimal path

$$a'_t = \lambda_t = u'_t + \delta E \lambda_{t+1}$$

where $a'_t$ and $u'_t$ are the marginal producer and user benefits, respectively. Thus, on the margin, the benefit received by producers this year from releasing one unit of water must equal the marginal benefit received by users this year from retaining the unit of water plus the benefits of having that unit available for either irrigation or recreation the following year.

The certainty-equivalent steady-state water level $s^*$, irrigation level $x^*$, and shadow price $\lambda^*$ solve the equation system

$$x^* = \bar{\epsilon}$$

$$a'(x^*) = \lambda^*$$

$$u'(s^* - x^*) = (1 - \delta) a'(x^*)$$

where $\bar{\epsilon}$ is mean annual rainfall. These conditions imply that the certainty-equivalent steady-state irrigation level and shadow price of water are not affected by the interest rate. The certainty-equivalent steady-state reservoir level, however, is affected by the interest rate. Totally differentiating the above equation system and making reasonable assumptions about the curvature of the benefit functions:

$$\frac{\partial s^*}{\partial r} = \frac{\delta^2 a'}{u''} < 0.$$

That is, as the interest rate rises, the certainty-equivalent steady-state reservoir level falls.

### 8.3.5 Optimal Monetary Policy

A monetary authority wishes to control the nominal interest rate $x$ in order to minimize the volatility of the inflation rate $s_1$ and the gross domestic product (GDP) gap $s_2$. Specifically, the authority wishes to minimize expected discounted stream of weighted squared deviations from zero targets

$$L(s) = \frac{1}{2} s^\top \Omega s$$

where $s$ is a 2x1 vector containing the inflation rate and the GDP gap and $\Omega$ is a 2x2 constant positive definite matrix of weights. The inflation rate and the GDP gap are a joint controlled linear Markov process

$$s_{t+1} = \alpha + \beta s_t + \gamma x + \epsilon$$

where $\alpha$ and $\gamma$ are 2x1 constant vectors, $\beta$ is a 2x2 constant matrix, and $\epsilon$ is a 2x1 random vector with mean zero. For political reasons, the nominal interest rate $x$ cannot be negative. What monetary policy minimizes the sum of current and expected future losses?

This is an infinite horizon, stochastic model with two state variables, the inflation rate $s_1$ and the GDP gap $s_2$, and one choice variable, the nominal inflation rate $x$, which is subject to the constraint $x \geq 0$. In order to formulate this problem as a maximization problem, one may posit a reward function that equals the negative of the loss function

$$f(s) = -L(s).$$

Given this assumption, the sum of current and expected future rewards $V(s)$ satisfies Bellman's equation

$$V(s) = \max_{0 \leq x} \{-L(s) + \delta E V(g(s, x, \epsilon))\}.$$

Given the structure of the model, one cannot preclude the possibility that the nonnegativity constraint on the optimal nominal interest rate will be binding in certain states. As such, the shadow price function $\lambda(s)$ is characterized by the Euler conditions

$$\delta \gamma^\top E \lambda(g(s, x, \epsilon) = \mu$$

$$\lambda(s) = -\Omega s + \delta \beta^\top E \lambda(g(s, x, \epsilon))$$

where the nominal interest rate $x$ and the long-run marginal reward $\mu$ from increasing the nominal interest rate must satisfy the complementarity condition

$$x \geq 0, \qquad \mu \leq 0, \qquad x > 0 \Longrightarrow \mu = 0.$$

It follows that along the optimal path

$$\delta\gamma^\top E_t\lambda_{t+1} = \mu_t$$

$$\lambda_t = -\Omega s_t + \delta\beta^\top E\lambda_{t+1}$$

$$x_t \geq 0, \qquad \mu_t \leq 0, \qquad x_t > 0 \Longrightarrow \mu_t = 0.$$

Thus, in any period, the nominal interest rate is reduced until either the long-run marginal reward or the nominal interest rate is driven to zero.

### 8.3.6 Production-Adjustment Model

A competitive price-taking firm wishes to manage production so as to maximize long run profits, given that production is subject to adjustment costs. In particular, if the firm produces a quantity $q$, it incurs production costs $c(q)$ and adjustment costs $0.5a(q-l)^2$, where $l$ is the preceding period's (lagged) production. The firm can sell any quantity it produces at the prevailing market price, which is an exogenous Markov process

$$p_{t+1} = g(p_t, \epsilon_{t+1}).$$

What production policy maximizes the value of the firm?

This is an infinite horizon, stochastic model with two state variables, the current market price $p$ and lagged production $l$, and one choice variable, production $q$, which is subject to the nonnegativity constraint $q \geq 0$. The current reward, short-run profits, equals revenue less production and adjustment costs

$$f(p, l, q) = p\,q - c(q) - 0.5a(q-l)^2.$$

The value $V(p, l)$ of the firm, given the market price $p$ and the previous period's production $l$, satisfies Bellman's equation

$$V(p, l) = \max_{0 \leq q}\{p\,q - c(q) - a(q-l) + \delta EV(g(p, \epsilon), q)\}.$$

Assuming a positive optimal production level in all states, the shadow price of lagged production $\lambda(p, l)$ will satisfy the Euler equilibrium conditions

$$p - c'(q) - a'(q-l) + \delta E\lambda(g(p, \epsilon), q) = 0$$

$$\lambda(p, l) = a'(q-l).$$

It follows that along the optimal path

$$p_t = c'_t + (a'_t - \delta Ea'_{t+1})$$

where $c'_t$ and $a'_t$ are the marginal production and adjustment costs in period $t$. Thus, price equals the marginal cost of production plus the net (current less future) marginal adjustment cost.

The certainty-equivalent steady-state production $q^*$ is obtained by assuming $p$ is fixed at its long-run mean $\bar{p}$:

$$\bar{p} = c'(q^*) + (1-\delta)a'(0).$$

### 8.3.7 Production-Inventory Model

A competitive price-taking firm wishes to manage production and inventories so as to maximize long-run profits. The firm begins each period with a predetermined stock of inventory $s$ and decides how much to produce $q$ and how much to store $x$, buying or selling the resulting difference $s+q-x$ on the open market at the prevailing price $p$. The firm's production and storage costs are given by $c(q)$ and $k(x)$, respectively, and the market price follows a purely exogenous Markov process

$$p_{t+1} = g(s_t, \epsilon_{t+1}).$$

What production policy and inventory policy maximizes the value of the firm?

This is an infinite horizon, stochastic model with two state variables, the current market price $p$ and beginning inventories $s$, and two choice variables, production $q$ and ending inventories $x$, both of which are subject to nonnegativity constraints $q \geq 0$ and $x \geq 0$. The current reward, short-run profits, equals net revenue from marketing sales or purchases, less production and storage costs:

$$f(p, s, q, x) = p(s+q-x) - c(q) - k(x).$$

The value $V(p, s)$ of the firm, given market price $p$ and beginning inventories $s$, satisfies Bellman's equation

$$V(p,s) = \max_{0 \leq q, 0 \leq x} \{p(s+q-x) - c(q) - k(x) + \delta E V(g(p,\epsilon), x)\}.$$

If production is subject to increasing marginal costs and $c'(0)$ is sufficiently small, then production will be positive in all states and the shadow price of beginning inventories $\lambda(p, s)$ will satisfy the Euler equilibrium conditions:

$$p = c'(q)$$

$$\delta E \lambda(g(p,\epsilon), x) - p - k'(x) = \mu$$

$$\lambda(p, s) = p$$

$$x \geq 0, \qquad \mu \leq 0, \qquad x > 0 \Longrightarrow \mu = 0.$$

It follows that along the optimal path,

$$p_t = c_t'$$

$$x_t \geq 0, \qquad E_t p_{t+1} - p_t - k_t' \leq 0, \qquad x > 0 \Longrightarrow E_t p_{t+1} - p_t - k_t' = 0.$$

where $p_t$ denotes the market price, $c_t'$ denotes the marginal production cost, and $k_t'$ denotes the marginal storage cost. Thus, the firm's production and storage decisions are independent. Production is governed by the conventional short-run profit maximizing condition that price equal the marginal cost of production. Storage, on the other hand, is entirely driven by intertemporal arbitrage profit opportunities. If the expected marginal profit from storing is negative, then no storage is undertaken. Otherwise, stocks are accumulated up to the point at which the marginal cost of storage equals the present value expected appreciation in the market price.

The certainty-equivalent steady-state obtains when $p$ is fixed at its long-run mean $\bar{p}$, in which case no appreciation can take place and optimal inventories will be zero. The certainty-equivalent steady-state production is implicitly defined by the short-run profit maximization condition.

### 8.3.8 Optimal Feeding

An livestock producer feeds his stock up to period $T$ and then sells it at the beginning of period $T+1$ at a fixed price $p$ per unit weight. Each period, the producer must determine how much grain $x$ to feed his livestock, given that grain sells at a constant unit cost $c$. The weight of the livestock at the beginning of each period is a controlled first-order deterministic process

$$s_{t+1} = g(s_t, x_t).$$

What feeding policy maximizes profit, given that the weight of the livestock in the initial period, $t = 0$, is $\bar{s}$?

This is an finite horizon, deterministic model with one state variable, the livestock weight at beginning of the period $s \in [\bar{s}, \infty)$, and one choice variable, the amount of feed purchased $x \in [0, \infty)$, which is subject to the constraint $x \geq 0$. The value of livestock weighing $s$ in period $t$ satisfies Bellman's equation

$$V_t(s) = \max_{x \geq 0} \{-cx + \delta V_{t+1}(g(s, x))\},$$

subject to the terminal condition

$$V_{T+1}(s) = ps.$$

If the marginal weight gain $g_x$ at zero feed is sufficiently large, the nonnegativity constraint of feed will never be binding. Under these conditions, the shadow price of livestock weight in period $t$, $\lambda_t(s)$, will satisfy the Euler equilibrium conditions:

$$\delta\lambda_{t+1}(g(s,x))g_x(s,x) = c$$

$$\lambda_t(s) = \delta\lambda_{t+1}(g(s,x))g_s(s,x)$$

subject to the terminal condition

$$\lambda_{T+1}(s) = p.$$

It follows that along the optimal path

$$\delta\lambda_{t+1}g_{x,t} = c$$

$$\lambda_t = \delta\lambda_{t+1}g_{s,t}$$

where $g_{x,t}$ and $g_{s,t}$ represent, respectively, the marginal weight gain from feed and the marginal decline in the livestock's ability to gain weight as it grows in size. Thus, the cost of feed must equal the value of the marginal weight gain. Also, the present valued shadow price grows at a rate that exactly counters the marginal decline in the livestock's ability to gain weight.

## 8.4 Linear-Quadratic Control

The linear-quadratic control problem is an unconstrained Markov decision model with a quadratic reward function

$$f(s,x) = F_0 + F_s s + F_x x + 0.5 s^\top F_{ss} s + s^\top F_{sx} x + 0.5 x^\top F_{xx} x$$

and a linear state transition function

$$g(s,x,\epsilon) = G_0 + G_s s + G_x x + \epsilon.$$

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 $\epsilon$ is assumed to have a mean of zero.

The linear-quadratic control model is of special importance because it is one of the few discrete time continuous state Markov decision models with a finite-dimensional solution. By a conceptually simple but algebraically burdensome induction proof

omitted here, one can show that the optimal policy and shadow price functions of the infinite horizon linear-quadratic control model are both linear in the state variable:

$$x(s) = \Gamma_0 + \Gamma_s s$$

$$\lambda(s) = \Lambda_0 + \Lambda_s s.$$

Here, $\Gamma_0$ is an $m$-by-1 vector, $\Gamma_s$ is an $m$-by-$n$ matrix, $\Lambda_0$ is an $n$-by-1 vector, and $\Lambda_s$ is an $n$-by-$n$ matrix.

The parameters $\Lambda_0$ and $\Lambda_s$ of the shadow price function are characterized by the nonlinear vector fixed point Riccati equations

$$\begin{aligned}
\Lambda_0 &= -[\delta {G_s}^\top \Lambda_s G_x + F_{sx}][\delta {G_x}^\top \Lambda_s G_x + {F_{xx}}^\top]^{-1}[\delta {G_x}^\top [\Lambda_s G_0 + \Lambda_0] + {F_x}^\top] \\
&\quad + \delta {G_s}^\top [\Lambda_s G_0 + \Lambda_0] + {F_s}^\top \\
\Lambda_s &= -[\delta {G_s}^\top \Lambda_s G_x + F_{sx}][\delta {G_x}^\top \Lambda_s G_x + {F_{xx}}^\top]^{-1}[\delta {G_x}^\top \Lambda_s G_s + {F_{sx}}^\top] \\
&\quad + \delta {G_s}^\top \Lambda_s G_s + F_{ss}.
\end{aligned}$$

These finite-dimensional fixed-point equations can typically be solved in practice using function iteration. The recursive structure of these equations allow one to first solve for $\Lambda_s$ by applying function iteration to the second equation, and then solve for $\Lambda_0$ by applying function iteration to the first equation. Once the parameters of the shadow price function have been computed, one can compute the parameters of the optimal policy via algebraic operations:

$$\Gamma_0 = -[\delta {G_x}^\top \Lambda_s G_x + {F_{xx}}^\top]^{-1}[\delta {G_x}^\top [\Lambda_s G_0 + \Lambda_0] + {F_x}^\top]$$

$$\Gamma_s = -[\delta {G_x}^\top \Lambda_s G_x + {F_{xx}}^\top]^{-1}[\delta {G_x}^\top \Lambda_s G_s + {F_{sx}}^\top]$$

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 depend only on the mean of the state shock, but not its variance or higher moments. 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 $\epsilon$ 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 non-binding or non-existent, and if the shocks have relatively 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 known to hold globally, or very nearly so.

## 8.5 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 the others. To simplify notation, we consider only infinite horizon games. The theory and methods developed, however, can be easily adapted to accommodate finite horizons.

The discrete time continuous state Markov $m$-agent game has the following structure: In every period, each agent $i$ observes the state of an economic process $s \in S$, takes an action $x_i \in X$, and earns a reward $f_i(s, x_i, x_{-i})$ that depends on the state of the process and both the action taken by the agent and the actions taken by the $m-1$ other agents $x_{-i}$. The state of the economic process is a jointly controlled Markov process. Specifically, the state of the economic process in period $t+1$ will depend on the state in period $t$, the actions taken by all $m$ agents in period $t$, and an exogenous random shock $\epsilon_{t+1}$ that is unknown in period $t$:

$$s_{t+1} = g(s_t, x_t, \epsilon_{t+1}).$$

As with static games, the equilibrium solution to a Markov game depends on the information available to the agents and the strategies they are assumed to pursue. We will limit discussion to noncooperative Markov perfect equilibria, that is, equilibria that yield a Nash equilibrium in every proper subgame. Under the assumption that each agent can perfectly observe the state of the process and knows the policies

followed by the other agents, a Markov perfect equilibrium is a set of $m$ policies of state-contingent actions $x_i^* : S \mapsto X$, $i = 1, 2, \ldots, m$, such that policy $x_i^*$ maximizes the present value of agent $i$'s current and expected future rewards, discounted at a per-period factor $\delta$, given that other agents pursue their policies $x_{-i}^*(\cdot)$. That is, for each agent $i$, $x_i^*(\cdot)$ solves

$$\max_{\{x(\cdot)\}} E_0 \sum_{t=0}^{T} \delta^t f_i(s_t, x_i(s_t), x_{-i}^*(s_t))$$

The Markov perfect equilibrium for the $m$-agent game is characterized by a set of $m$ simultaneous Bellman equations

$$V_i(s) = \max_{x \in X_i(s)} \left\{ f_i(s, x, x_{-i}^*(s)) + \delta E_\epsilon V_i(g(s, x, x_{-i}^*(s), \epsilon)) \right\}.$$

whose unknowns are the value functions $V_i(\cdot)$ and optimal policies $x_i^*(\cdot)$, $i = 1, 2, \ldots, m$ of the different agents. Here, $V_i(s)$ denotes the maximum current and expected future rewards that can be earned by agent $i$, given that other agents pursue their optimal strategies.

### 8.5.1 Capital-Production Game

Consider two infinitely-lived firms that produce perishable goods that are close substitutes (say, donuts and bagels). Each firm $i$ begins period $t$ with a predetermined capital stock $k_i$ and must decide how much to produce $q_i$. Its production cost $c_i(q_i, k_i)$ depends on both the quantity produced and the capital stock. Prices are determined by short-run market clearing conditions (Cournot competition). More specifically, firm $i$ receives price $p_i = P_i(q_1, q_2)$ that depends both on its output and the output of its competitor. The firm must also decide how much to invest in capital. Specifically, if the firm invests in new capital $x_i$, it incurs a cost $h_i(x_i)$ and its capital stock at the beginning of the following period will be $(1 - \xi)k_i + x_i$ where $\xi$ is the capital depreciation rate. What are the two firm's optimal production and investment policies?

This is an infinite horizon, deterministic 2-agent dynamic game with two state variables, the capital stocks of the two producers, $k_1$ and $k_2$. Each agent $i$ has two decision variables, production $q_i$ and investment $x_i$, which are subject to the nonnegativity constraints $q_i \geq 0$ and $x_i \geq 0$. His current reward, net revenue, equals $P_i(q_1, q_2)q_i - c_i(q_i, k_i) - h_i(x_i)$. The Markov perfect equilibrium for the production-capital game is represented by a pair of Bellman equations, one for each firm, which take the form

$$V_i(k_1, k_2) = \max_{q_i \geq 0, x_i \geq 0} \{P_i(q_1, q_2)q_i - c_i(q_i, k_i) - h_i(x_i) + \delta E_\epsilon V_i(\hat{k}_1, \hat{k}_i)\}$$

where $\hat{k}_i = (1 - \xi)k_i + x_i$.

### 8.5.2 Risk-Sharing Game

Consider two infinitely-lived agents who must make consumption-investment decisions. Each period, each agent $i$ begins with a predetermined level of wealth $s_i$, of which an amount $x_i$ is invested, and the remainder is consumed, yielding an utility $u_i(s_i - x_i)$. Agent $i$'s wealth at the beginning of period $t+1$ is determined entirely by his investment in period $t$ and an income shock $\epsilon_{t+1}$, which is unknown at the time the investment decision is made. More specifically, wealth follows a controlled Markov process

$$s_{it+1} = g_i(x_{it}, \epsilon_{it+1}).$$

Suppose now that the two agents co-insure against exogenous income risks by agreeing to share their wealth in perpetuity. Specifically, the agents agree that, at the beginning of any given period $t$, the wealthier of the two agents will transfer a certain proportion $\sigma$ of the wealth differential to the poorer agent. Under this scheme, agent $i$'s wealth in period $t+1$, after the transfer, will equal

$$s_{it+1} = (1-\sigma)g_i(x_{it}, \epsilon_{it+1}) + \sigma g_j(x_{jt}, \epsilon_{jt+1}).$$

where $j \neq i$. If the wealth transfer is enforceable, but agents are free to consume and invest freely, moral hazard will arise. In particular, both agents will have incentives to shirk investment in favor of current consumption when co-insured. How will insurance affect the agents' behavior, and for what initial wealth states $s_1$ and $s_2$ and share parameter $\sigma$ will both agents be willing to enter into the insurance contract? How does the correlation in the wealth shocks affect the value of the insurance contract?

This is an infinite horizon, stochastic 2-agent dynamic game with two state variables, the wealth levels of the two agents $s_1$ and $s_2$. Each agent $i$ has a single decision variable, his investment $x_i$, which is subject to the constraint $0 \leq x_i \leq \hat{s}_i$. His current period reward, current utility, equals $u_i(\hat{s}_i - x_i)$. The Markov perfect equilibrium for the redistribution game is represented by a pair of Bellman equations, one for each agent, which take the form

$$V_i(s_1, s_2) = \max_{0 \leq x_i \leq \hat{s}_i} \{u_i(\hat{s}_i - x_i) + \delta E_\epsilon V_i(\hat{s}_1, \hat{s}_2)\},$$

where $\hat{s}_i = (1-\sigma)g_i(x_i, \epsilon_i) + \sigma g_j(x_j, \epsilon_j)$. Here, $V_i(s_1, s_2)$ denotes the maximum expected lifetime utility that can be obtained by agent $i$.

### 8.5.3 Marketing Board Game

Suppose that two countries are the sole producers of a commodity and that, 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, attempting to maximize the present value of their own current and expected future income from commodity sales. More specifically, the marketing board in country $i$ begins each period with a pre-determined supply $s_i$ of the commodity, of which it exports a quantity $q_i$ and stores the remainder $s_i - q_i$ at a total cost $c_i(s_i - q_i)$. The world market price will depend on the total amount exported by both countries, $p = p(q_1 + q_2)$. The supplies available in the two countries at the beginning period $t+1$ are given by

$$s_{it+1} = x_{it} + y_{it}$$

where new production in both countries, $y_{1t}$ and $y_{2t}$, are assumed to be exogenous and independently and identically distributed over time. What are the optimal export strategies for the two marketing boards?

This is an infinite horizon, stochastic 2-agent dynamic game with two state variables, the beginning supplies in the two counties $s_1$ and $s_2$. The marketing board for country $i$ has a single decision variable, the export level $q_i$, which is subject to the constraint $0 \leq x_i \leq s_i$. Country $i$'s current reward, net income, equals $p(q_1 + q_2)q_i - c_i(s_i - q_i)$. The Markov perfect equilibrium for the marketing board game is captured by a pair of Bellman equations, one for each marketing board, which take the form

$$V_i(s_1, s_2) = \max_{0 \leq q_i \leq s_i} \{p(q_1 + q_2)q_i - c_i(s_i - q_i) + \delta E_y V_i(s_1 - q_1 + y_1, s_2 - q_2 + y_2)\}.$$

Here, $V_i(s_1, s_2)$ denotes the maximum current and expected future income that can be earned by marketing board $i$, given that marketing board $j$ remains committed to its export policy.

## 8.6 Rational Expectations Models

We now examine dynamic stochastic models of economic systems in which arbitrage-free equilibria are enforced through the collective, decentralized actions of atomistic dynamically optimizing agents. We assume that agents are rational in the sense that their expectations are consistent with the implications of the model as whole. Examples of phenomenon that may be studied in a rational expectations framework include asset returns in a pure exchange economy, pricing of primary commodities, and agricultural production subject to price controls.

We limit attention to dynamic models of the following form: At the beginning of period $t$, an economic system emerges in a state $s_t$. Agents observe the state of the system and, by pursuing their individual objectives, produce a systematic response $x_t$

governed by an equilibrium condition that depends on expectations of the following period's state and action

$$f(s_t, x_t, E_t h(s_{t+1}, x_{t+1})) = 0.$$

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 $\epsilon_{t+1}$ that is realized only after the system responds at time $t$:

$$s_{t+1} = g(s_t, x_t, \epsilon_{t+1}).$$

In many applications, the equilibrium condition $f = 0$ admits a natural arbitrage interpretation. In these instances, $f_i > 0$ indicates activity $i$ generates profits on the margin, so that agents have a collective incentive to increase $x_i$; $f_i < 0$ indicates that activity $i$ generates loses on the margin, so that agents have a collective incentive to decrease $x_i$. An arbitrage-free equilibrium exists if and only if $f = 0$.

The state space $S \in \Re^n$, which contains the states attainable by the economic system, and the response space $X \in \Re^m$, which contains the admissible system responses, are both closed convex nonempty sets. The functions $f : \Re^{n+m+p} \mapsto \Re^m$, $g : \Re^{n+m+n} \mapsto \Re^n$, and $h : \Re^{n+m} \mapsto \Re^p$ are continuously differentiable and the per-period discount factor $\delta$ is less than one. The exogenous random shocks $\epsilon_t$ are identically distributed over time, mutually independent, and independent of past states and responses. In some instances, the range of admissible responses may vary with the state of the process $s$. In such cases, the restricted response space will be denoted $X(s)$. The stipulation that the response in any period depends only the expectations of only the subsequent period's state and response is more general than first appears. By introducing new accounting variables, responses can be made dependent on expectations of states and responses further in the future.

The primary task facing an economic analyst is to derive the rational expectations equilibrium system response $x = x(s)$ for each state $s$. The response function $x(\cdot)$ is characterized implicitly as the solution to a functional equation

$$f\Big(s, x(s), E_\epsilon\Big[h(g(s, x(s), \epsilon), x(g(s, x(s), \epsilon)))\Big]\Big) = 0.$$

The equilibrium condition takes a different form when the system response is constrained. Suppose, for example, that responses are subject to bounds of the form

$$a(s) \leq x \leq b(s),$$

where $a : S \mapsto X$ and $b : S \mapsto X$ are continuous functions of the state $s$. In these instances, the arbitrage condition takes the form

$$f(s_t, x_t, E_t h(s_{t+1}, x_{t+1})) = \mu_t$$

where $x_t$ and $\mu_t$ satisfy the complementarity condition

$$a(s_t) \leq x \leq b(s_t), \qquad x_{ti} > a_i(s_t) \Longrightarrow \mu_{ti} \geq 0, \qquad x_{ti} < b_i(s) \Longrightarrow \mu_{ti} \leq 0.$$

Here, $\mu_t$ is am $m$-vector whose $i^{th}$ element, $\mu_{ti}$, measures the marginal benefit from activity $i$. In equilibrium, $\mu_{ti}$ must be nonpositive if $x_{ti}$ is less than its upper bound, for otherwise agents can gain by increasing activity $i$; similarly, $\mu_{ti}$ must be nonnegative if $x_{ti}$ is greater than its lower bound, for otherwise agents can gain by reducing activity $i$. And if $x_{ti}$ is neither at its upper or lower bound, $\mu_{ti}$ must be zero to ensure the absence of arbitrage opportunities from revising the level of activity $i$.

### 8.6.1 Asset Pricing Model

Consider a pure exchange economy in which a representative infinitely-lived agent allocates wealth between immediate consumption and investment. Wealth is held in shares, $s_t$, of claims that pay a dividend of $d_t$ units of a consumption good per share with the current price of a share being $p_t$. The representative agent's objective is choose consumption levels, $c_t$, to maximize discounted expected utility subject to an intertemporal budget constraint. The budget constraint stipulates that the current value of shares purchased in this period must equal the total dividends paid on beginning of period shares less consumption:

$$p_t(s_{t+1} - s_t) = d_t s_t - c_t$$

which implies the state transition equation

$$s_{t+1} = s_t + (d_t s_t - c_t)/p_t.$$

Thus the agent solves

$$V(s_t, d_t, p_t) = \max_{c_t \in [0, d_t s_t]} E_0 \left[ \sum_{t=0}^{\infty} \delta^t U(c_t) \right] \text{ s.t. } s_{t+1} = s_t + (d_t s_t - c_t)/p_t.$$

Under mild regularity conditions, the agent's dynamic optimization problem has an unique solution that satisfies the first-order Euler condition

$$U'(c_t)p_t = \delta E_t \Big[ U'(c_{t+1})(p_{t+1} + d_{t+1}) \Big]$$

(see exercise 8.21). 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 dividend tomorrow.

In a representative agent economy, all agents behave in identical fashion and hence no shares are bought or sold (autarky). Furthermore, if we normalize the total number

of shares to equal the population size then the consumption level will equal $c_t = d_t$. The model is closed by assuming that process $d_t$ is an exogenous Markov process

$$d_{t+1} = g(d_t, \epsilon_{t+1}).$$

The asset pricing model is an infinite horizon, stochastic model that may be formulated with one state variable, the dividend level $d$, one response variable, the asset price $p$, and one equilibrium condition

$$U'(d_t)p_t - \delta E_t[U'(d_{t+1})(p_{t+1} + d_{t+1})] = 0,$$

which asserts that the expected marginal utility from saving is zero.

A solution to the rational expectations asset pricing model is a function $p(d)$ that gives the equilibrium asset price $p$ in terms of the exogenous dividend level $d$. From the dynamic equilibrium conditions, the asset return function is characterized by the functional equation

$$U'(d)p(d) - \delta E_\epsilon\Big[U'\Big(g(d,\epsilon)\Big)\Big(p(g(d,\epsilon)) + g(d,\epsilon)\Big)\Big] = 0.$$

In the notation of the general model, with $s = d$ and $x = p$,

$$h(s,x) = U'(s)(x+s),$$

and

$$f(s,x,Eh) = U'(s)x - \delta Eh.$$

### 8.6.2 Competitive Storage

Consider a market for a storable primary commodity. Each period $t$ begins with a predetermined supply of the commodity $s_t$, of which an amount $q_t$ is sold to consumers at a market clearing price $p_t = P(q_t)$ and the remainder $x_t$ is stored. Supply at the beginning of the following period is the sum of current carryout and exogenous new production $y_{t+1}$, which is uncertain in period $t$:

$$s_{t+1} = x_t + y_{t+1}.$$

Competitive storers seeking to maximize expected profits guarantee that profit opportunities are fully exploited in equilibrium. In particular,

$$\delta E_t[p_{t+1}] - p_t - c = \mu_t$$

$$x_t \geq 0, \qquad \mu_t \leq 0. \qquad x_t > 0 \Longrightarrow \mu_t = 0$$

where $\mu_t$ equals profit from storing one unit of the commodity. Whenever expected profits are positive, storers increase stockholdings, raising the current market price and lowering the expected future price, until profits are eliminated. Conversely, whenever expected profits are negative, storers decrease stockholdings, lowering the current market price and raising the expected future price, until either expected losses are eliminated or stocks are depleted.

The commodity storage model is an infinite horizon, stochastic model. The model may be formulated with one state variable, the supply $s$ available at the beginning of the period, one response variable, the storage level $x$, and one equilibrium condition

$$\delta E_t \left[ P\left( s_{t+1} - x_{t+1} \right) \right] - P(s_t - x_t) - c = \mu_t$$

$$x_t \geq 0, \qquad \mu_t \leq 0, \qquad x_t > 0 \Longrightarrow \mu_t = 0.$$

A solution to the commodity storage model formulated in this fashion is a function $x(\cdot)$ that gives the equilibrium storage in terms of the available supply. From the dynamic equilibrium conditions, the equilibrium storage function is characterized by the functional complementarity condition

$$\delta E_y[P(x(s) + y - x(x(s) + y))] - P(s - x(s)) - c = \mu(s)$$

$$x(s) \geq 0, \qquad \mu(s) \leq 0, \qquad x(s) > 0 \Longrightarrow \mu(s) = 0.$$

In the notation of the general model

$$h(s, x) = P(s - x)$$

$$g(s, x, y) = s + y - x$$

and

$$f(s, x, Eh) = \delta Eh - P(s - x) - c.$$

The commodity storage model also admits an alternate formulation with the market price $p$ as the sole response variable. In this formulation, the equilibrium condition takes the form

$$\delta E_t[p_{t+1}] - p_t - c = \mu_t$$

$$p_t \geq P(s_t), \qquad \mu_t \leq 0, \qquad p_t > P(s_t) \Longrightarrow \mu_t = 0.$$

A solution to the commodity storage model formulated in this fashion is a function $\lambda(\cdot)$ that gives the equilibrium market price in terms of the available supply. From

the dynamic equilibrium conditions, the equilibrium price function is characterized by the functional complementarity condition

$$\delta E_y[\lambda(s - D(\lambda(s) + y)] - \lambda(s) - c = \mu(s)$$

$$\lambda(s) \geq P(s), \qquad \mu(s) \leq 0, \qquad \lambda(s) > P(s) \Longrightarrow \mu(s) = 0.$$

where $D = P^{-1}$ is the demand function. In the notation of the general model

$$h(s, p) = p$$

$$g(s, p, y) = s + y - D(p)$$

and

$$f(s, p, Eh) = \delta Eh - p - c.$$

The two formulations are mathematically equivalent. The equilibrium price function may be derived from the equilibrium storage function through the relation

$$\lambda(s) = P(s - x(s)).$$

The equilibrium storage function may be derived from the equilibrium price function through the relation

$$x(s) = s - D(\lambda(s)).$$

### 8.6.3 Government Price Controls

Consider a market for an agricultural commodity in which the government is committed to maintaining a minimum price through the management of a public buffer stock. In particular, the government stands ready to purchase and store unlimited quantities of the commodity at a fixed price $p^*$ in times of excess supply and to sell any quantities in its stockpile at the price $p^*$ in times of short supplies. Assume that there is no private stockholding.

Each year $t$ begins with a predetermined supply of the commodity $s_t$, of which an amount $q_t$ is sold to consumers at a market clearing price $p_t = P(q_t)$ and the remainder $x_t$ is stored by the government. Supply at the beginning of the following year is the sum of government stocks and new production, which equals the acreage planted by producers $a_t$ times an exogenous per-acre yield $y_{t+1}$, which is uncertain in year $t$

$$s_{t+1} = x_t + a_t y_{t+1}.$$

In making planting decisions, producers maximize expected profits by equating expected per-acre revenue to the marginal cost of production, which is a function of the acreage planted

$$\delta E_t p_{t+1} y_{t+1} = c(a_t).$$

The government price control model is an infinite horizon, stochastic model with two state variables, the supply $s$ available at the beginning of the period and the yield $y$, two response variables, the acreage planted $a$ and government storage $x$, and two equilibrium conditions

$$\delta E_t P(s_{t+1} - x_{t+1}) y_{t+1} - c(a_t) = 0,$$

which asserts that the marginal expected profit from planting is zero, and

$$x_t \geq 0, \qquad p^* \geq P(s_t - x_t), \qquad x_t > 0 \Longrightarrow p^* = P(s_t - x_t),$$

which asserts that the government will store the quantities necessary to enforce the price floor, but will not store otherwise. A solution to the government price control model are a pair of functions $x(\cdot)$ and $a(\cdot)$ that give government storage and acreage planting in terms of available supply. From the dynamic equilibrium conditions, the equilibrium government storage and acreage planting functions are characterized by the simultaneous functional complementarity problem

$$\delta E_y\Big[P\Big(x(s) + y - x(x(s) + y)\Big)\Big] - P(s - x(s)) - c(x(s)) = 0$$

and

$$x(s) \geq 0, \qquad p^* \geq P(s - x(s)), \qquad x(s) > 0 \Longrightarrow p^* = P(s - x(s)).$$

In the notation of the general model the state variable is $(s, y)$, the response variable is $(x, a)$ and the shock process is $y$. The expectation function is

$$h(s, y, x, a) = P(s - x)y$$

and the equilibrium function is

$$f(s, x, Eh) = \begin{bmatrix} p^* - P(s - x) \\ \delta Eh - c(a) \end{bmatrix}$$

with $a$ unbounded and $x \in [0, \infty]$.

**Exercises**

8.1. 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 $0 < \beta < 1$.

A firm operating without regard to environmental consequences produces at its profit maximizing level $q_t = \alpha_1$. Suppose that the social welfare, accounting for environmental damage, 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 $\tau$ that will induce the firm to produce at the steady-state socially optimal production level $q^*$.

8.2. 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 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 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 $\alpha_0$ rises)? How do they change if the harvest cost rises (i.e., if $\beta_0$ rises)?

8.3. 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 $K$ dollars. The period after cutting, $S = 0$.

(a) Formulate and interpret Bellman's equation.

(b) What conditions characterize the certainty equivalent steady-state?

8.4. Repeat the last exercise but assume now that the per unit price of cut timber satisfies

$$\ln(P_{t+1}) = \bar{p} + \alpha\Big(\ln(P_t) - \bar{p}\Big) + e_{t+1},$$

where $e \sim i.i.d.\ N(0, \phi^2)$ and is independent of $S$.

8.5. 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 rate?

8.6. 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: $x_t = x(s_t)$?

8.7. 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 $\tau_t$ is tax paid per unit of capital. The firm's production function, adjustment 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) Identify the state and action variables, the reward function, and the transition function of this problem.

(b) Write Bellman's functional equation. What does the value function represent?

(c) Assuming an internal solution, derive the Euler conditions and interpret them. What does the shadow price function represent?

(d) What effect does an increase in the base tax rate, $\tau$, have on output in the long run.

(e) What effect does an increase in the discount factor, $\delta$, have on output in the long run.

8.8. Consider the optimal growth model in section 8.3.1. Find and sign $\dfrac{\partial s^*}{\partial \gamma}$, $\dfrac{\partial x^*}{\partial \gamma}$, and $\dfrac{\partial \lambda^*}{\partial \gamma}$.

8.9. Consider the renewable resource model in section 8.3.2. 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?

8.10. Hogs breed at a rate $\beta$. 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.

8.11. 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 $\beta$ 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 action variables, the reward function, and the transition function of this problem.

(b) Write Bellman's functional equation. What does the value function represent?

(c) Derive the Euler conditions and interpret them. What does the shadow price 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.

8.12. 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, $k_0$.

(a) Identify the state and action variables, the reward function, and the transition function of this problem.

(b) Write Bellman's functional equation. What does the value function represent?

(c) Derive the Euler conditions and interpret them. What does the shadow price function represent?

(d) Show that the value function has the form

$$V(k_t) = A + B\ln(k_t)$$

and that the optimal decision rule for this problem is

$$k_{t+1} = Cy_t;$$

find the values for $A$, $B$, and $C$.

8.13. 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) Identify the state and action variables, the reward function, and the transition function of this problem.

(b) Write Bellman's functional equation. What does the value function represent?

(c) Derive the Euler conditions and interpret them. What does the derivative of value function represent?

(d) What conditions characterize the steady-state optimal solution?

8.14. A corn producer's net per-acre revenue in year $t$ is given by

$$c_t = py_t - cx_t - wl_t$$

where $p$ is the unit price of corn (\$/bu.), $y_t$ is the corn yield (bu./acre), $c$ is the unit cost of fertilizer (\$/lb.), $x_t$ is the amount of fertilizer applied (lbs./acre), $w$ 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) Identify the state and action variables, the reward function, and the transition function of this problem.

(b) Write Bellman's functional equation. What does the value function represent?

(c) Derive the Euler conditions and interpret them. What does the derivative of value function represent?

(d) What conditions characterize the steady-state optimal solution?

8.15. The role of commodity storage in intertemporal allocation has often been controversial. In particular, the following claims have often been made: a) Competitive storers, in search of speculative profits, tend to hoard a commodity—that is, they collectively store more than is socially optimal, and b) 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.

8.16. 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.

8.17. Show that the competitive storage model in section 8.6.2 can be formulated with the equilibrium storage function as the sole unknown. Hint: Write the arbitrage storage condition in the form $f(s_t, x_t, E_t h(x_{t+1})) = 0$ for some appropriately defined function $h$.

8.18. Show that the competitive storage model of section 8.6.2 can be recast as a dynamic optimization problem. In particular, formulate a dynamic optimization problem in which a hypothetical social planner maximizes the discounted expected sum of consumer surplus less storage costs. Derive the Euler conditions to show that, under a suitable interpretation, they are identical to the rational expectations equilibrium conditions of the storage model.

8.19. Consider the production-inventory model of section 8.3.7. Show that the value function is of the form $V(p, s) = ps + W(p)$ where $W$ is the solution to a Bellman functional equation. Can you derive general conditions under which one can reduce the dimensionality of a Bellman equation?

8.20. Consider the monetary policy model of section 8.3.5. Derive the certainty-equivalent steady-state inflation rate, GDP gap, nominal interest rate, and shadow prices under the simplifying assumption that the nominal interest rate is unconstrained.

8.21. Demonstrate that the problem

$$V(s_t, d_t, p_t) = \max_{c_t \in [0, d_t s_t]} E_0 \left[ \sum_{t=0}^{\infty} \delta^t U(c_t) \right] \text{ s.t. } s_{t+1} = s_t + (d_t s_t - c_t)/p_t.$$

leads to the Euler condition

$$\delta E_t[U'(c_{t+1})(p_{t+1} + d_{t+1})] = U'(c_t)p_t.$$

# Chapter 9

# Discrete Time Continuous State Dynamic Models: Methods

This chapter discusses numerical methods for solving discrete time continuous state dynamic economic models. Such models give rise to functional equations whose unknowns are entire functions defined on a subset of Euclidean space. For example, the unknown of Bellman's equation

$$V(s) = \max_{x \in X(s)} \{f(s,x) + \delta E_\epsilon V(g(s,x,\epsilon))\}$$

is the value function $V(\cdot)$. And the unknown of a rational expectations equilibrium condition

$$f(s,x(s),E_\epsilon h(g(s,x(s),\epsilon),x(g(s,x(s),\epsilon)))) = 0$$

is the response function $x(\cdot)$.

In most applications, these functional equations lack known closed form solution and can only be solved approximately using computational methods. Among the computational methods available, linear-quadratic approximation and space discretization historically have been popular among economists due to the relative ease with which they can be implemented. However, in most applications, these methods either provide unacceptably poor approximations or are computationally inefficient.

In recent years, economists have begun to experiment with projection methods pioneered by physical scientists. Among the various projection methods available, the collocation method is the most useful for solving dynamic models in Economics and Finance. In most applications, the collocation method is flexible, accurate, and numerically efficient. It can also be developed directly from basic numerical integration, approximation, and rootfinding techniques.

The collocation method employs a conceptually straightforward strategy to solve functional equations. Specifically, the unknown function is approximated using a linear combination of $n$ known basis functions whose $n$ coefficients are fixed by requiring the approximant to satisfy the functional equation, not at all possible points of the domain, but rather at $n$ prescribed points called the collocation nodes. The collocation method effectively replaces an infinite-dimensional functional equation with a finite-dimensional nonlinear equation that can be solved using standard numerical rootfinding, fixed-point, and complementarity techniques.

Unfortunately, the widespread applicability of the collocation method to economic and financial models has been hampered by the absence of publicly available general purpose computer code. We address this problem by developing computer routines that perform the essential computations for a broad class of dynamic economic and financial models. Below, the collocation method is developed in greater detail for single- and multiple-agent decision Bellman equations and rational expectations models. Application of the method is illustrated with a variety of examples.

## 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 model to be replaced with a discrete state and action decision model that closely resembles it. The resulting discrete state and action model is then solved using the dynamic programming methods discussed in Chapter 7. To "discretize" the state space of a continuous state Markov decision problem, one partitions the state space $S$ 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 regions $X_1, X_2, \ldots, X_m$. 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 state discrete action Markov decision problem. The transition probabilities of the discrete state discrete action space problem are computed by integrating with respect to the density of the random shock:

$$P(s_{i'}|s_i, x_j) = \Pr[g(s_i, x_j, \epsilon) \in S_{i'}].$$

When the state and action spaces are bounded intervals on the real line, 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, 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

$$P(s_{i'}|s_i, x_j) = \Pr[s_{i'} - w_s/2 \leq g(s_i, x_j, \epsilon) \leq s_{i'} + w_s/2].$$

Another popular method for solving dynamic optimization models is linear-quadratic approximation. Linear-quadratic approximation calls for the state 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 numerically. Typically, the linear and quadratic approximants of $g$ and $f$ are constructed by forming the first- and second-order Taylor expansions around the certainty-equivalent steady-state. When passing to the linear-quadratic approximation, any constraints on the action, 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 $\epsilon^*$ denotes the mean shock, the certainty-equivalent steady-state state $s^*$, optimal action $x^*$, and shadow price $\lambda^*$ are characterized by the nonlinear equation system:

$$f_x(s^*, x^*) + \delta\lambda^* g_x(s^*, x^*, \epsilon^*) = 0$$

$$\lambda^* = f_s(s^*, x^*) + \delta\lambda^* g_s(s^*, x^*, \epsilon^*)$$

$$s^* = g(s^*, x^*, \epsilon^*).$$

Typically, the nonlinear equation may be solved for the steady-state values of $s^*$, $x^*$, and $\lambda^*$ using standard nonlinear equation methods. In one-dimensional state and action models, the conditions can often be solved analytically. Here, $f_x$, $f_s$, $g_x$, and $g_s$ denote partial derivatives whose dimensions are $1 \times m$, $1 \times n$, $n \times m$, and $n \times n$, respectively, where $n$ and $m$ are the dimensions of the state and action spaces, respectively. Here, the certainty-equivalent steady-state shadow price $\lambda^*$ is expressed as a $1 \times n$ row vector.

The second step is to replace the state transition function $g$ and the reward function $f$, respectively, with their first- and second-order Taylor series approximants expanded around the certainty-equivalent steady-state:

$$\begin{aligned} f(s,x) \quad \approx\ & f^* + f_s^*(s-s^*) + f_x^*(x-x^*) + 0.5(s-s^*)^\top f_{ss}^*(s-s^*) \\ & + (s-s^*)^\top f_{sx}^*(x-x^*) + 0.5(x-x^*)^\top f_{xx}^*(x-x^*) \end{aligned}$$

$$g(s,x,\epsilon) \approx g^* + g_s^*(s-s^*) + g_x^*(x-x^*).$$

Here, $f^*$, $g^*$, $f_s^*$, $f_x^*$, $g_s^*$, $g_x^*$, $f_{ss}^*$, $f_{sx}^*$, and $f_{xx}^*$ are the values and partial derivatives of $f$ and $g$ evaluated at the certainty-equivalent steady-state. If $n$ and $m$ are the dimensions of the state and action spaces, respectively, then the orders of these vectors and matrices are as follows: $f^*$ is a constant, $f_s^*$ is $1 \times n$, $f_x^*$ is $1 \times m$, $f_{ss}^*$ is $n \times n$, $f_{sx}^*$ is $n \times m$, $f_{xx}^*$ is $m \times m$, $g^*$ is $n \times 1$, $g_s^*$ is $n \times n$, and $g_x^*$ is $n \times m$.

The shadow price and optimal policy functions of the resulting linear-quadratic control problem will be linear. Specifically:

$$x(s) = x^* + \Gamma(s-s^*)$$

$$\lambda(s) = \lambda^* + \Lambda(s-s^*).$$

The slope matrices of the policy and shadow price functions, $\Lambda$ and $\Gamma$, are characterized by the nonlinear vector fixed point equations

$$\Lambda = -[\delta g_s^{*\top}\Lambda g_x^* + f_{sx}^*][\delta g_x^{*\top}\Lambda g_x^* + f_{xx}^{*\top}]^{-1}[\delta g_x^{*\top}\Lambda g_s^* + f_{sx}^{*\top}] + \delta g_s^{*\top}\Lambda g_s^* + f_{ss}^*$$

$$\Gamma = -[\delta g_x^{*\top}\Lambda g_x^* + f_{xx}^{*\top}]^{-1}[\delta g_x^{*\top}\Lambda g_s^* + f_{sx}^{*\top}].$$

These fixed point equations can usually be solved using numerically by function iteration, typically with initial guess $\Lambda = 0$, or, if the problem is one dimensional, analytically by applying the quadratic formula. In particular, 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:

$$\Lambda = [f_{ss}^* g_x^{*2} - 2f_{ss}^* f_{xx}^* g_s^* g_x^* + f_{xx}^* g_s^{*2} - f_{xx}^*/\delta]/g_x^{*2}$$

## 9.2 The Collocation Method

In order to describe the collocation method for solving continuous state Markov decision models, we will limit our discussion to infinite-horizon models with one-dimensional state and action spaces and univariate shocks. The presentation generalizes to models with higher dimensional states, actions, and shocks, but at the expense of cumbersome additional notation required to track the different dimensions.[1]

[1] The routines included in the Compecon library accompanying the book admit higher dimensional states, actions, and shocks.

Consider, then, Bellman's equation for an infinite horizon discrete time continuous state dynamic decision problem

$$V(s) = \max_{x \in X(s)} \{f(s,x) + \delta E_\epsilon V(g(s,x,\epsilon))\}.$$

Assume that the state space is a bounded interval of the real line, $S = [s_{min}, s_{max}]$, and the actions either are discrete or are continuous and subject to simple bounds $a(s) \leq x \leq b(s)$ that are continuous functions of the state. Further assume that the reward function $f(s,x)$ and state transition function $g(s,x,\epsilon)$ are twice continuously differentiable functions of their arguments.

To compute an approximate solution to Bellman's equation via collocation, one employs the following strategy: First, write the value function approximant as 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, fix the basis function coefficients $c_1, c_2, \ldots, c_n$ 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 polynomial and spline approximation schemes. The best choice of basis-node scheme is application specific, and often depends on the curvature of the value and policy functions.

The collocation strategy replaces the Bellman functional equation with a system of $n$ nonlinear equations in $n$ unknowns. Specifically, to compute the value function approximant, or more precisely, to compute the $n$ coefficients $c_1, c_2, \ldots, c_n$ in its basis representation, one must solve the nonlinear equation system

$$\sum_j c_j \phi_j(s_i) = \max_{x \in X(s_i)} \{f(s_i,x) + \delta E_\epsilon \sum_{j=1}^{n} c_j \phi_j(g(s_i,x,\epsilon))\}.$$

The nonlinear equation system may be compactly expressed in vector form as the *collocation equation*

$$\Phi c = v(c).$$

Here, $\Phi$, 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 *collocation function*, is the 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 collocation function evaluated at a particular vector of basis coefficients $c$ yields a vector whose $i^{th}$ entry is the value obtained by solving the optimization problem embedded in Bellman's equation at the $i^{th}$ collocation node, replacing the value function appearing in the optimand with the approximant $\sum_j c_j \phi_j$.

In principle, the collocation equation may be solved using any nonlinear equation solution method. For example, one may write the collocation equation as a fixed-point problem $c = \Phi^{-1} v(c)$ and employ function iteration, which uses 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 collocation function $v$ at $c$. The typical element of $v'$ may be computed by applying the Envelope Theorem to the optimization problem in the definition of $v(c)$. Specifically,

$$v'_{ij}(c) = \frac{\partial v_i}{\partial c_j}(c) = \delta E_\epsilon \phi_j(g(s_i, x_i, \epsilon))$$

where $x_i$ is the optimal argument in the maximization problem producing $v_i(c)$. As a variant to Newton's method one could also employ a quasi-Newton method to solve the collocation equation.[2]

If the model is stochastic, one must compute expectations in a numerically practical way. Regardless of the quadrature scheme selected, the continuous random variable $\epsilon$ 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 collocation function $v$ takes the specific form

$$v_i(c) = \max_{x \in X(s_i)} \{f(s_i, x) + \delta \sum_{k=1}^{m} \sum_{j=1}^{n} w_k c_j \phi_j(g(s_i, x, \epsilon_k))\}.$$

[2]The Newton update rule is equivalent to $c \leftarrow [\Phi - v'(c)]^{-1} f$, where $f$ is the $n$ by 1 vector of optimal rewards at the state nodes. This is identical to the "policy iteration" rule commonly used in discrete state dynamic programming.

and its Jacobian takes the form

$$v'_{ij}(c) = \delta \sum_{k=1}^{m} w_k \phi_j(g(s_i, x_i, \epsilon_k)).$$

Let us now consider the practical steps that must be taken to implement the collocation method in a computer programming environment. Below, we outline the key operations using the Matlab vector processing language, presuming access to the function approximation and numerical quadrature routines contained in the Compecon library. The necessary steps can be implemented in virtually any other vector processing or high-level algebraic programming language, with a level of difficulty that will depend mainly on the availability of the required elementary approximation and quadrature routines.

Consider first a dynamic decision model with a discrete action space in which the possible actions are identified with the first `p` positive integers. The initial steps in any implementation of the collocation method are to specify the basis functions that will be used to express the value function approximant and to specify the collocation nodes at which the Bellman equation will be required to hold exactly. These steps may be executed using the Compecon library routines `fundefn`, `funnode`, and `funbas`, which are discussed in Chapter 6:

```
fspace = fundefn('cheb',n,smin,smax);
s      = funnode(fspace);
Phi    = funbas(fspace);
```

Here, it is presumed that the analyst has previously specified the lower and upper endpoints of the state interval, `smin` and `smax`, and the number of basis functions and collocation nodes `n`. After execution, `fspace` is a structured variable that contains the information needed to well-define the approximation basis, `s` is the `n` by `1` vector of standard collocation nodes associated with the basis, and `Phi` is the `n` by `n` collocation matrix associated with the basis. In this specific example, the Chebychev polynomial basis functions and collocation nodes are used to form the value function approximant via collocation.

Next, a numerical routine must be coded to evaluate the collocation function and its derivative at an arbitrary basis coefficient vector. A simple version of such a routine for discrete choice models would have a calling sequence of the form

```
[v,x,vjac] = vmax(s,c).
```

Here, on input, `s` is an `n` by `1` vector of collocation nodes and `c` is an `n` by `1` vector of basis coefficients. On output, `v` is an `n` by `1` vector of optimal values at the collocation

nodes, `x` is an `n` by `1` vector of associated optimal actions at the nodes, and `vjac` is an `n` by `n` Jacobian of the collocation function at `c`.

Given the collocation nodes `s`, collocation matrix `Phi`, and collocation function routine `vmax`, and given an initial guess for the basis coefficient vector `c`, the collocation equation may be solved either by function iteration

```
for it=1:maxit
  cold = c;
  [v,x] = vmax(s,c);
  c = Phi\v;
  if norm(c-cold)<tol, break, end;
end
```

or by Newton iteration

```
for it=1:maxit
  cold = c;
  [v,x,vjac] = vmax(s,c);
  c = cold - [Phi-vjac]\[Phi*c-v];
  if norm(c-cold)<tol, break, end;
end
```

Here, `tol` and `maxit` are iteration control parameters set by the analyst, specifying the convergence tolerance and the maximum number of iterations. The Matlab operator is used to perform the linear solve.

The main challenge in implementing the collocation method for a general class of dynamic optimization problems is coding the routine `vmax` that solves the optimization problem embedded in Bellman's equation at the collocation nodes and returns the collocation function values and derivatives. A simple routine that performs this optimizations for the discrete choice model is as follows:[3]

```
function [v,x,vjac] = vmax(s,c)
  Ev = 0;
  for i=1:p
    x = i*ones(n,1);
    f(:,i) = ffunc(s,x);
    for k=1:m
      g = gfunc(s,x,e(k));
```

[3]For clarity, the code omits several bookkeeping operations and programming tricks that accelerate execution. Operational versions of `vmax` that efficiently handle arbitrary dimensional state and actions spaces are included with the Compecon library routine `dpsolve`.

```
    Ev(:,i) = Ev(:,i) + w(k)*funeval(c,fspace,g);
  end
end
[v,x] = max(f+delta*Ev,[],2); vjac = 0;
for k=1:K
  g = gfunc(s,x,e(k));
  vjac = vjac + delta*w(k)*funbas(fspace,g);
end
```

This routine assumes that the analyst has coded separate ancillary routines `ffunc` and `gfunc` that return the rewards and state transitions specific to the dynamic decision model being solved. The routine `ffunc` accepts `n` by `1` vectors of state nodes `s` and actions `x` and returns an `n` by `1` vector `f` of associated reward function values. The routine `gfunc` accepts `n` by `1` vectors of state nodes `s` and actions `x` and a particular value of the shock `e` and returns an `n` by `1` vector `g` of associated state transition function values.

The routine `vmax` begins with the execution of a series of loops, one for each possible action. The loops produce `n` by `p` matrices `f` and `Ev` containing, respectively, the current reward and value expected next period associated with the `n` state nodes (rows) and the `p` possible actions (columns). The value expected next period is computed by looping over all `K` possible realizations of the discrete shock and forming the probability weighted sum of values next period (here, `e(k)` and `w(k)` are the $k^{th}$ shock and its probability). For each realization of the shock, the state next period `g` is computed and passed to the routine `funeval`, which returns next period's value using value function approximant associated with the coefficient vector `c`.

By construction, `f+delta*Ev` is an `n` by `p` matrix whose entries give for each state node (row) and action (column) the current reward `f` plus the expected value next period `Ev` discounted at the rate `delta`. The maximum value in each row of `f+delta*Ev` and the associated column index are the optimal value and action associated with the corresponding state node. In this implementation of `vmax`, the Matlab vector maximization routine `max` is used to perform the column-wise maximization in one call, yielding `n` by `1` vectors `v` and `x` that contain the optimal values and actions associated with the `n` state nodes. The Jacobian `vjac` of the collocation function is computed by executing a loop over all `K` possible realizations of the discrete shock. For each realization of the shock, the state next period `g` is computed and passed to the Compecon library routine `funbas`, which returns the basis function values at that state node.

Consider now a dynamic decision model with a continuous, rather than a discrete, action space. The steps required to solve a continuous choice model using collocation are identical to those required to solve a discrete choice model, with the exception

of the optimization routine `vmax`. As with a discrete choice model, the analyst must specify the basis functions and collocation nodes and code a numerical routine to evaluate the collocation function and its derivative at an arbitrary basis coefficient vector. Armed with these elements and given initial guesses for the basis coefficient vector `c` and optimal actions `x`, the collocation equation may be solved either by function iteration or by Newton iteration, just as before.

The only difference between discrete and continuous choice implementations of the collocation method lies with the routine `vmax` that solves the optimization problem embedded in Bellman's equation. A routine that performs the optimization for the continuous choice model by iteratively solving the associated Karush-Kuhn-Tucker complementarity conditions is as follows:

```
function [v,x,vjac] = vmax(s,x,c)
 [xl,xu] = bfunc(s);
 for it=1:maxit
   [f,fx,fxx] = ffunc(s,x);
   Ev=0; Evx=0; Evxx=0;
   for k=1:K
     [g,gx,gxx] = gfunc(s,x,e(k));
     vn     = funeval(c,fspace,g);
     vnder1 = funeval(c,fspace,g,1);
     vnder2 = funeval(c,fspace,g,2);
     Ev   = Ev   + w(k)*vn;
     Evx  = Evx  + w(k)*vnder1.*gx;
     Evxx = Evxx + w(k)*(vnder1.*gxx + vnder2.*gx.^2);
   end
   v    = f + delta*Ev;
   delx = -(fx+delta*Evx)./(fxx+delta*Evxx);
   delx = min(max(delx,xl-x),xu-x);
   x    = x + delx;
   if norm(delx)<tol, break, end;
 end vjac = 0;
 for k=1:K
   g = gfunc(s,x,e(k));
   phinext = funbas(fspace,g);
   vjac = vjac + delta*w(k)*phinext;
 end
```

This routine assumes that the analyst has coded separate ancillary routines `bfunc`, `ffunc`, and `gfunc` that compute the bounds, rewards, and state transitions specific to the dynamic decision model being solved. The routine `bfunc` accepts an `n` by `1` vector

of states `s` and returns `n` by `1` vectors `xl` and `xu` of associated lower and upper bounds on the actions. The routine `ffunc` accepts `n` by `1` vectors of states `s` and actions `x` and returns `n` by `1` vectors `f`, `fx`, and `fxx` of associated reward function values, first derivatives, and second derivatives. The routine `gfunc` accepts `n` by `1` vectors of states `s` and actions `x` and a particular value of the shock `e` and returns `n` by `1` vectors `g`, `gx`, and `gxx` of associated state transition function values, first derivatives, and second derivatives.

The continuous action routine `vmax` begins by computing the lower and upper bounds `xl` and `xu` on the actions at the state nodes. The routine then executes a series of Newton iterations that sequentially update the actions `x` at the state nodes `s` until the Karush-Kuhn-Tucker conditions of the optimization problem embedded in Bellman's equation are satisfied to a specified tolerance `tol`. With each iteration, the standard Newton step `delx` is computed and subsequently shortened, if necessary, to ensure that the updated action `x+delx` remains within the bounds `xl` and `xu`. The standard Newton step `delx` is the negative of the ratio of the second and third derivatives of the Bellman optimand with respect to the action, `fx+delta*Evx` and `fxx+delta*Evxx`. Here, `fx` and `fxx` are the first and second derivatives of the reward function, `Evx` and `Evxx` are the first and second derivatives of the expected value next period, and `delta` is the discount rate.

In order to compute the expected value next period and its derivatives, a loop is executed over all `K` possible realizations of the discrete shock and probability weighted sums are formed (here, `e(k)` and `w(k)` are the $k^{th}$ shock and its probability). For each realization of the shock, the state next period `g` and its first and second derivatives with respect to the action, `gx` and `gxx`, are computed. The state next period is passed to the library routine `funeval`, which computes next period's value and its derivatives using the value function approximant that is identified with the current coefficient vector `c`. The Chain Rule is then used to compute the derivatives of the expected value.

Once convergence is achieved and the optimal value and action at the state nodes have been determined, the Jacobian `vjac` of the collocation function is computed. The Jacobian is a `n` by `n` matrix whose representative $ij^{th}$ entry is the discounted expectation of the $j^{th}$ basis evaluated at the following period's state, given the current state is the $i^{th}$ state node. To compute the Jacobian, a loop is executed over all `K` possible realizations of the discrete shock. For each realization of the shock, the state next period `g` is computed and passed to the Compecon library routine `funbas`, which evaluates the basis functions at that state.

Once the collocation equation has apparently been solved, the analyst should perform a diagnostic test to assure that the computed value function approximant solves Bellman's equation to an acceptable degree of accuracy over the entire interpolation

interval. In order to perform this test, define the residual function

$$R_c(s) = \max_{x \in X(s)} \{f(s,x) + \delta E_\epsilon \sum_{j=1}^{n} c_j \phi_j(g(s,x,\epsilon))\} - \sum_j c_j \phi_j(s),$$

which measures the difference between the right and left sides of the Bellman equation at arbitrary states $s$ when the value function is replaced with its approximant $\sum_j c_j \phi_j$. If the approximant provides an exact solution to Bellman's equation, the residual will be zero throughout the interpolation interval. Of course, in practice, the residual of an approximant will typically be nonzero, except at the collocation nodes where it is zero by design. However, if the residual function between the collocation nodes is close to zero, the value function approximant is deemed acceptable. Otherwise, if large residuals obtain, the model should be solved again with more collocation nodes, or different basis functions, or a revised interpolation interval, until the norm of the residual function is reduced to acceptable levels.

In practice, the easiest way to assess the Bellman equation approximation error is to plot the residual function on a fine grid of states spanning the interpolation interval. The residual function may be computed at any vector of states using the routines `vmax` and `funeval`. For example, the approximation residual of a discrete action model approximant may be checked as follows:

```
nres  = 500;
sres  = nodeunif(smin,smax,nres);
resid = vmax(sres,c) - funeval(c,fspace,sres);
plot(sres,resid)
```

Here, the Compecon library routine `nodeunif` is used to generate a vector `sres` of 500 equally spaced states spanning the interpolation interval. The residual `resid` is then computed and plotted at the equally-spaced states. Notice that, to perform compute the residual, `vmax` is evaluated at the residual evaluation points, not the collocation nodes. However, careful inspection of the code above reveals that `vmax` is designed to solve the optimization problem embedded in Bellman's equation at an arbitrary vector of states, not just the collocation nodes.

There are two common causes of poor residuals. First, the value function may exhibit discontinuous derivatives along a boundary separating regions of the state space where the solution exhibits qualitatively different characteristics. In a discrete action model, each region may correspond to a family of states at which a given discrete action is optimal. In a continuous action model, the regions may separate states according to whether action constraints are binding or nonbinding. This existence of discontinuous second derivatives in the value function creates difficulties for approximation schemes based on Chebychev polynomials and cubic splines, both of which

are twice continuously differentiable. In particular, the residuals will tend to be much larger in magnitude near the kink points. If the kink points are known analytically, which is rarely the case in practice, the residual error can often be reduced by the choosing collocation nodes so as to include the kink points. Another remedy that is often effective is to by simply increase the number of basis functions and collocation nodes, though this many not be computationally practical when the state space is higher-dimensional.

Another possible cause of poor residuals is extrapolation beyond the interpolation interval. Since interpolants can provide highly inaccurate approximations when evaluated outside the specified interpolation interval, one should check whether this occurs within the routine `vmax` at the final solution. The test can be performed by enumerating all values that can be realized by the state transition function at all possible state nodes, corresponding optimal actions, and shocks, and checking that the value remains between `smin` and `smax`. In Matlab, this can be executed with the following commands:

```
for k=1:K;
  g = gfunc(s,x,e(k));
  if any(g<smin), disp('Warning: reduce smin  '), end;
  if any(g>smax), disp('Warning: increase smax'), end;
end
```

If the Bellman residuals are poor and there are attempts to extrapolate within `vmax` are detected, the minimum and/or maximum state should be extended and the model should be solved again. The interpolation interval should be repeatedly adjusted in this manner until extrapolation beyond the interval no longer occurs or the residual function becomes acceptably small.

## 9.3 Postoptimality 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 this behavior changes with variations model parameters or assumptions. Given a dynamic economic model, we typically characterize the model's solution in one of two ways. Steady-state analysis examines the long-run tendencies of the optimized process, abstracting from the initial state and the 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 computed 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 iteratively as follows: $s_{t+1} = g(s_t, x(s_t))$. Given the path of the state variable $s_t$, it is then usually 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 computed optimal policy function $x(s)$, the transition function $g(s, x, \epsilon)$, an initial state $s_0$, and a pseudorandom sequence of $\epsilon_t$, a representative path may be generated iteratively as follows: $s_{t+1} = g(s_t, x(s_t), \epsilon_{t+1})$.

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. The expected path is typically smooth and converges to a steady-state mean value.

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.

## 9.4 Computational Examples

### 9.4.1 Asset Replacement

Consider the asset replacement model of Section 8.2.1 assuming that an asset produces $q(a) = (50 - 2.5a - 2.5a^2)$ units of output in its $a^{th}$ period of operation, up to period $\bar{a} = 10$, and that the output price is $p = 2$. Further assume that the replacement cost $k$ is an exogenous continuous-valued first-order Markov process $k_{t+1} = g(k_t, \epsilon_{t+1}) = \bar{k} + \gamma(k_t - \bar{k}) + \epsilon_{t+1}$ where $\epsilon_t$ is i.i.d. normal$(0, \sigma^2)$.

The collocation method calls for the analyst to select $n$ basis functions $\phi_j$ and $n$ collocation nodes $(k_i, a_i)$, and form the value function approximant $V \approx \sum_{j=1}^{n} c_j \phi_j$ whose coefficients $c_j$ solve the collocation equation

$$\sum_{j=1}^{n} c_j \phi_j(k_i, a_i) =$$

$$\max\{p\ q(a_i) + \delta \sum_{j=1}^{n} w_k c_j \phi_j(\hat{k}_{ik}, a_i + 1), p\ q(0) - k_i + \delta \sum_{j=1}^{n} w_k c_j \phi_j(\hat{k}_{ik}, 1)\}.$$

where $\hat{k}_{ik} = g(k_i, \epsilon_k)$ and where $\epsilon_k$ and $w_k$ represent Gaussian quadrature nodes and weights for the normal shock. In practice, one may solve the asset replacement model using Compecon library routines as follows:[4]

**Step 1** Code model function file:

```
function out = mfdp01(flag,s,x,e,price,cbar,gamma);
switch flag
case 'f'; % REWARD FUNCTION
  out = price*(50-2.5*s(:,2)-2.5*s(:,2).^2).*(1-x)...
      +(price*50-s(:,1)).*x;
case 'g'; % STATE TRANSITION FUNCTION
  out(:,1) = cbar + gamma*(s(:,1)-cbar) + e;
  out(:,2) = min(s(:,2)+1,10).*(1-x) + x;
end
```

[4]Functioning Matlab code for this example is contained in the Compecon library demonstration file `demdp01`.

The model function file returns the values of the reward and transition functions at arbitrary vectors of states `s`, actions `x`, and shocks `e`. Passing the flag `'f'` returns the reward function value `f` and passing the flag `'g'` returns the transition function value `g`.

**Step 2** Enter model parameters:

```
delta   = 0.9;
price   = 2.0;
cbar    = 100;
gamma   = 0.5;
sigma   = 15;
```

Here, the discount factor, output price, long-run mean replacement cost, replacement cost autoregression coefficient, and standard deviation of replacement cost shock are specified, respectively.

**Step 3** Discretize shock:

```
m = 5;
[e,w] = qnwnorm(m,0,sigma^2);
```

Here, the normal replacement cost shock is discretized using a five node Gaussian quadrature scheme.

**Step 4** Specify basis functions and collocation nodes:

```
n = 60;
cmin = 30;
cmax = 190;
fspace = fundefn('lin',n,cmin,cmax,[],[1:10]');
snodes = funnode(fspace);
s = gridmake(snodes);
```

Here, a 60-function finite difference basis on the interval $[30, 190]$ is used to approximate the value function along its continuous dimension (replacement cost) and the associated collocation nodes are used to formulate the collocation equation.

**Step 5** Construct the action space:

```
x = [0;1];
```

Here, the action space is dichotomous.

**Step 6** Pack model structure:

```
model.func = 'mfdp01';
model.discount = delta;
model.e = e;
model.w = w;
model.actions = x;
model.discretestates = 2;
model.params = {price cbar gamma};
```

Here, `model` is a structured variable whose fields contain the elements of the dynamic decision model. The first field contains the name of the model function file `'mfdp01'`; the remaining fields contain the discount factor `delta`, shock values `e`, shock probabilities `w`, the action space `x`, the index of the discrete state (age) `discretestates`, and model function parameters `params`, respectively.

**Step 7** Provide judicious guesses for values at the collocation nodes:

```
vinit = zeros(size(s,1),1);
```

Here, the value function is initialized to zero.

**Step 8** Solve the decision model:

```
[c,s,v,x,resid] = dpsolve(model,fspace,snodes,vinit);
```

The Compecon routine `dpsolve` accepts as input the model structure `model`, basis functions `fspace`, collocation nodes `snodes`, and initial guess for the value function `vinit`. It then solves the collocation equation, returning as output the basis coefficients `c`, and the optimal values `v`, optimal actions `x`, and Bellman equation residuals `resid` at a refined state grid `s`.

**Step 8** Perform postoptimality analysis. Figure 9.1a gives the value of the firm as a function of the asset replacement costs for different asset ages. For any given asset age, the value of the firm is downward sloping and kinked at the critical replacement cost, below which the asset is replaced. Figure 9.1b gives the Bellman equation residual for the finite difference basis approximant as a function of the asset replacement costs for different asset ages. The residual exhibits noticeable errors at the critical replacement costs, which can be expected due to the discontinuous derivatives of the value function at those points.

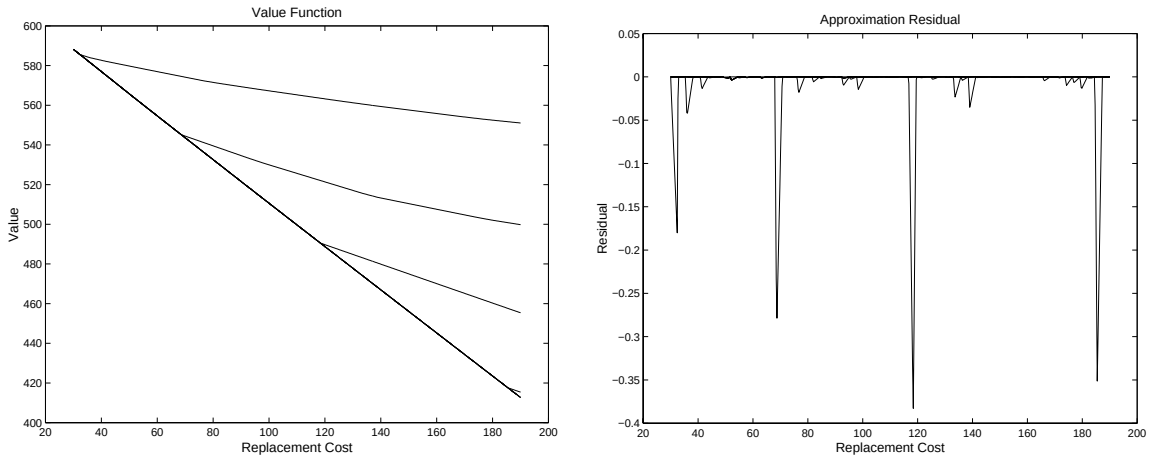


Figure 9.1: Solution to Asset Replacement Model

### 9.4.2 Timber Cutting

Consider the timber cutting model of Section 8.2.2 assuming that the stand biomass $s$ grows at a deterministic rate $s_{t+1} = K + \exp^{-\alpha}(s_t - K)$. Further assume a constant profit contribution per unit of biomass $p$ and constant cost of replanting $C$.

The collocation method calls for the analyst to select $n$ basis functions $\phi_j$ and $n$ collocation nodes $(s_i)$, and form the value function approximant $V \approx \sum_{j=1}^{n} c_j \phi_j$ whose coefficients $c_j$ solve the collocation equation

$$\sum_{j=1}^{n} c_j \phi_j(s_i) = \max\{\delta \sum_{j=1}^{n} c_j \phi_j(g(s_i))), P\ s_i - C + \delta \sum_{j=1}^{n} c_j \phi_j(0)\}.$$

In practice, one may solve the timber cutting model using Compecon library routines as follows:[5]

**Step 1** Code model function file:

```
function out = mfdp02(flag,s,x,e,price,C,K,alpha);
switch flag
case 'f';
  out = (price*s-C).*x;
case 'g';
```

[5]Functioning Matlab code for this example is contained in the Compecon library demonstration file `demdp02`.

```
  out = ((K+exp(-alpha)*(s-K))).*(1-x);
end
```

The model function file returns the values of the reward and transition functions at arbitrary vectors of states `s`, actions `x`, and shocks `e`. Passing the flag `'f'` returns the reward function value `f` and passing the flag `'g'` returns the transition function value `g`.

**Step 2** Enter model parameters:

```
delta =  0.95;
price =  1;
C     =  0.2;
K     =  0.5;
alpha =  0.1;
```

Here, the discount factor, output price, replanting cost, carrying capacity, and speed of mean reversion are specified, respectively.

**Step 3** Specify basis functions and collocation nodes:

```
n =  350;
fspace = fundefn('lin',n,0,K);
snodes = funnode(fspace);
```

Here, a 350-function finite difference basis on the interval $[0, K]$ and the associated standard collocation nodes are used to formulate the collocation equation.

**Step 4** Construct the action space:

```
x = [0;1];
```

Here, the action space is dichotomous.

**Step 5** Pack model structure:

```
model.func = 'mfdp02';
model.discount = delta;
model.actions = x;
model.params = {price C K alpha};
```

Here, `model` is a structured variable whose fields contain the elements of the dynamic decision model. The first field contains the name of the model function file `'mfdp02'`; the remaining fields contain the discount factor `delta`, the action space `x`, and model function parameters `params`, respectively.

**Step 6** Provide judicious guesses for values at the collocation nodes:

```
vinit = zeros(size(snodes));
```

Here, the value function is initialized to zero.

**Step 7** Solve the decision model:

```
[c,s,v,x,resid] = dpsolve(model,fspace,snodes,vinit);
```

The Compecon routine `dpsolve` accepts as input the model structure `model`, basis functions `fspace`, collocation nodes `snodes`, and initial guess for the value function `vinit`. It then solves the collocation equation, returning as output the basis coefficients `c`, and the optimal values `v`, optimal actions `x`, and Bellman equation residuals `resid` at a refined state grid `s`.

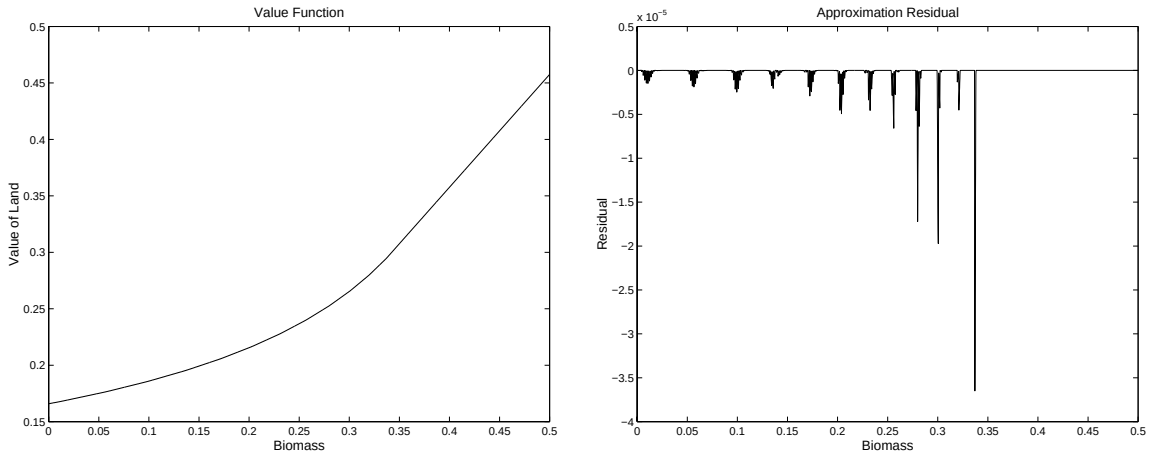


Figure 9.2: Solution to Timber Cutting Model

**Step 8** Perform postoptimality analysis. Figure 9.2a gives the value of the stand as a function of the biomass. Figure 9.2b gives the Bellman equation residual for the finite difference basis approximant.

### 9.4.3 Optimal Economic Growth

Consider the optimal economic growth model of Section 8.3.1 assuming a social benefit function $u(c) = c^{1-\alpha}/(1-\alpha)$, an aggregate production function $f(x) = x^{\beta}$, and an i.i.d.

lognormal$(0, \sigma^2)$ production shock $\epsilon$. The collocation method calls for the analyst to select $n$ basis functions $\phi_j$ and $n$ collocation nodes $s_i$, and form the value function approximant $V \approx \sum_{j=1}^{n} c_j \phi_j$ whose coefficients $c_j$ solve the collocation equation

$$\sum_{j=1}^{n} c_j \phi_j(s_i) = \max_{0 \le x \le s_i} \{(s_i - x)^{1-\alpha}/(1-\alpha) + \delta \sum_{k=1}^{m} \sum_{j=1}^{n} w_k c_j \phi_j(\gamma x + \epsilon_k x^{\beta})\}$$

where $\epsilon_k$ and $w_k$ represent Gaussian quadrature nodes and weights for the lognormal shock. In practice, one may solve the optimal economic growth model using Compecon library routines as follows:[6]

**Step 1** Code model function file:

```
function [out1,out2,out3] = mfdp07(flag,s,x,e,alpha,beta,gamma);
switch flag
case 'b'; % BOUND FUNCTION
  out1 = zeros(size(s));
  out2 = s;
case 'f'; % REWARD FUNCTION
  out1 = ((s-x).^(1-alpha))/(1-alpha);
  out2 = -(s-x).^(-alpha);
  out3 = -alpha*(s-x).^(-alpha-1);
case 'g'; % STATE TRANSITION FUNCTION
  out1 = gamma*x + e.*x.^beta;
  out2 = gamma + beta*e.*x.^(beta-1);
  out3 = (beta-1)*beta*e.*x.^(beta-2);
end
```

The model function file returns the values and derivatives of the bound, reward, and transition functions at arbitrary vectors of states `s`, actions `x`, and shocks `e`. Passing the flag `'b'` returns the lower and upper bounds on the action `xl` and `xu`; passing the flag `'f'` returns the reward function value and its first and second derivatives with respect to the action, `f`, `fx`, and `fxx`; and passing the flag `'g'` returns the transition function value and its first and second derivatives with respect to the action, `g`, `gx`, and `gxx`.

**Step 2** Enter model parameters:

[6] Functioning Matlab code for this example is contained in the Compecon library demonstration file `demdp07`.

```
delta =  0.9;
alpha =  0.2;
beta  =  0.5;
gamma =  0.9;
sigma =  0.1;
```

Here, the discount factor, utility function parameter, production elasticity, capital survival rate, and production shock volatility are specified, respectively.

**Step 3** Discretize shock:

```
m = 3;
[e,w] = qnwlogn(m,0,sigma^2);
```

Here, the lognormal production shock is discretized using a three node Gaussian quadrature scheme.

**Step 4** Specify basis functions and collocation nodes:

```
n = 10;
smin = 5;
smax = 10;
fspace = fundefn('cheb',n,smin,smax);
snodes = funnode(fspace);
```

Here, the first ten Chebychev polynomials on the interval $[5, 10]$ and the corresponding standard Chebychev nodes are selected to serve as basis functions and collocation nodes.

**Step 5** Pack model structure:

```
model.func = 'mfdp07';
model.discount = delta;
model.e = e;
model.w = w;
model.params = {alpha beta gamma};
```

Here, `model` is a structured variable whose fields contain the elements of the dynamic decision model. The first field contains the name of the model function file `'mfdp07'`; the remaining fields contain the discount factor `delta`, shock values `e`, shock probabilities `w`, and model function parameters `params`, respectively.

**Step 6** Provide judicious guesses for values and actions at the collocation nodes:

```
estar = 1;
xstar = ((1-delta*gamma)/(delta*beta))^(1/(beta-1));
sstar = gamma*xstar + xstar^beta;
[vlq,xlq] = lqapprox(model,snodes,sstar,xstar,estar);
```

Here, the the certainty-equivalent steady-state shock, action, and state are computed analytically and passed to the Compecon library routine `lqapprox`, which returns the linear-quadratic approximation values and actions at the collocation nodes. The linear-quadratic approximation is used to initialize the collocation algorithm.

**Step 7** Solve the decision model:

```
[c,s,v,x,resid] = dpsolve(model,fspace,snodes,vlq,xlq);
```

The Compecon routine `dpsolve` accepts as input the model structure `model`, basis functions `fspace`, collocation nodes `snodes`, and initial guesses for the value function `vlq` and optimal actions `xlq`. It then solves the collocation equation, returning as output the basis coefficients `c`, and the optimal values `v`, optimal actions `x`, and Bellman equation residuals `resid` at a refined state grid `s`.

**Step 8** Perform postoptimality analysis. Figure 9.3a gives optimal investment as a percent of wealth computed using both Chebychev collocation and linear-quadratic approximation. The Chebychev collocation approximant is upward sloping and the linear-quadratic approximant is downward sloping, indicating a qualitative difference between the two. Figure 9.3b gives the Bellman equation residual for the Chebychev approximant. The residual possesses zeros at the collocation nodes by design and exhibits very nearly equal oscillations between the nodes, a property that is typical of Chebychev residuals when the underlying model is smooth and effectively unconstrained. In this example, a ten degree Chebychev approximation was sufficient to solve the Bellman equation to a residual error of order $2 \times 10^{-10}$, approximately seven orders of magnitude more accurate than the linear-quadratic approximant, whose residual is not drawn. These results suggest that linear-quadratic approximation can yield globally inaccurate solutions even when the underlying model is smooth and the constraints on the actions are not binding.

Figure 9.3c gives the expected path followed by the wealth level over time, beginning from a wealth level of 5. The expected path was computed by performing Monte Carlo simulations involving 2000 replications of twenty years in duration each, using the Compecon library routine `dpsimul`:

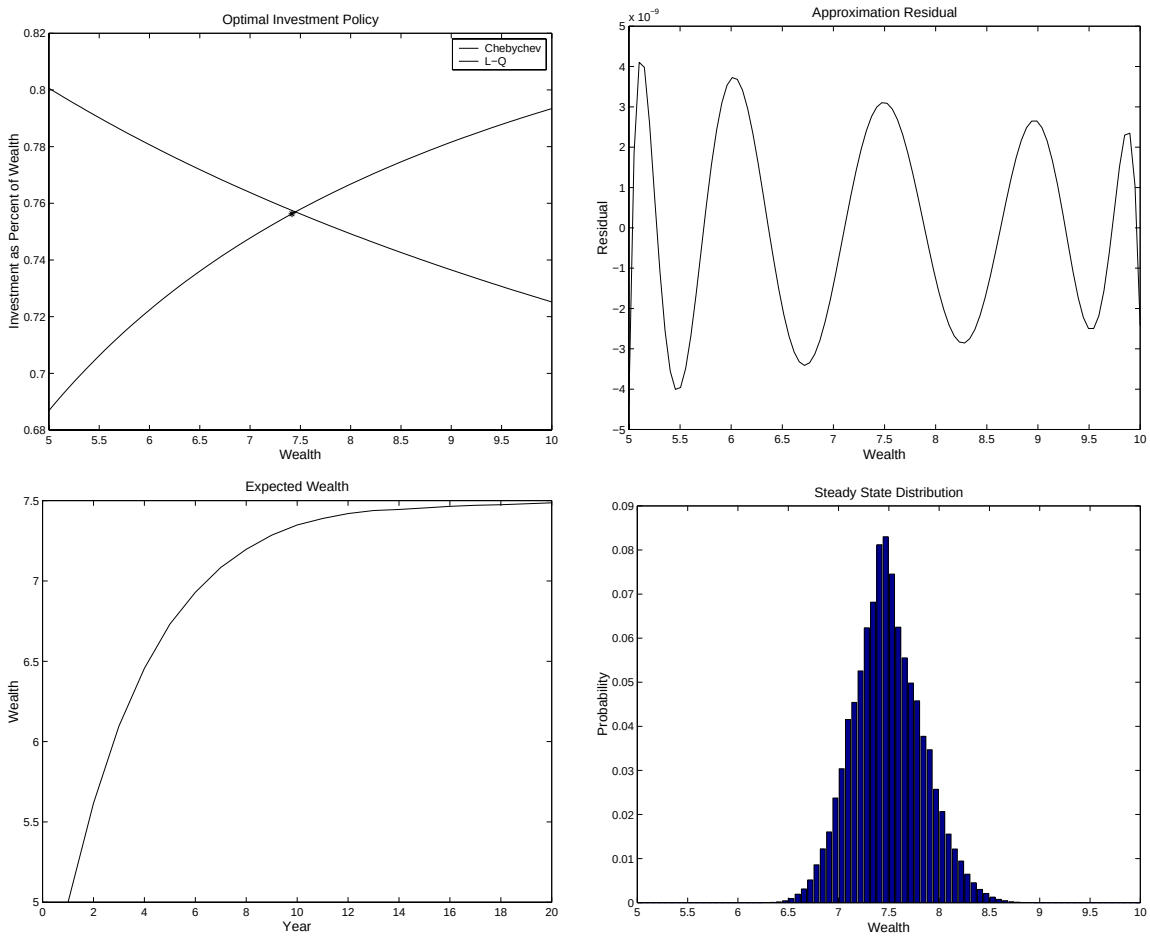


Figure 9.3: Solution to Optimal Economic Growth Model

```
nyrs = 20;
nrep = 2000;
sinit = 5*ones(nrep,1);
[spath,xpath] = dpsimul(model,sinit,nyrs,s,x);
```

As seen in this figure, expected wealth rises at a declining rate, converging asymptotically to a steady-state value of approximately 7.5. Figure 9.3d gives the steady-state distribution of the wealth level. The distribution, represented as an 80 bin histogram, was computed using using the Compecon library routine `dpstst` with a smoothing parameter of 5:

```
nsmooth = 5;
```

```
nbin = 80;
[ss,pi,xx] = dpstst(model,nsmooth,nbin,s,x);
```

As seen in this figure, the steady-state distribution is essentially bell-shaped with a mean of approximately 7.5, which is consistent with the Monte-Carlo state path simulations.

### 9.4.4 Public Renewable Resource Management

Consider the public renewable resource management model of Section 8.3.2 assuming an inverse demand function $p(x) = x^{-\gamma}$, a constant cost of harvest $k$, and a deterministic state transition function $g(s,x) = \alpha(s-x) - 0.5\beta(s-x)^2$. The collocation method calls for the analyst to select $n$ basis functions $\phi_j$ and $n$ collocation nodes $s_i$, and form the value function approximant $V \approx \sum_{j=1}^{n} c_j \phi_j$ whose coefficients $c_j$ solve the collocation equation

$$\sum_{j=1}^{n} c_j \phi_j(s_i) = \max_{0 \le x \le s_i} \{\frac{x^{1-\gamma}}{1-\gamma} - k\ x + \delta \sum_{j=1}^{n} c_j \phi_j(\alpha(s_i - x) - 0.5\beta(s_i - x)^2)\}.$$

In practice, one may solve the public renewable resource management model using Compecon library routines as follows:[7]

**Step 1** Code model function file:

```
function [out1,out2,out3] = mfdp08(flag,s,x,e,alpha,beta,gamma,cost);
 switch flag
 case 'b'; % BOUND FUNCTION
   out1 = zeros(size(s));
   out2 = s;
 case 'f'; % REWARD FUNCTION
   out1 = (x.^(1-gamma))/(1-gamma)-cost*x;
   out2 = x.^(-gamma)-cost;
   out3 = -gamma*x.^(-gamma-1);
 case 'g'; % STATE TRANSITION FUNCTION
   out1 = alpha*(s-x) - 0.5*beta*(s-x).^2;
   out2 = -alpha + beta*(s-x);
   out3 = zeros(size(s))-beta;
 end
```

[7] Functioning Matlab code for this example is contained in the Compecon library demonstration file `demdp08`.

The model function file returns the values and derivatives of the bound, reward, and transition functions at arbitrary vectors of states `s` and actions `x`. Passing the flag `'b'` returns the lower and upper bounds on the action `xl` and `xu`; passing the flag `'f'` returns the reward function value and its first and second derivatives with respect to the action, `f`, `fx`, and `fxx`; and passing the flag `'g'` returns the transition function value and its first and second derivatives with respect to the action, `g`, `gx`, and `gxx`.

**Step 2** Enter model parameters:

```
delta = 0.9;
alpha = 4.0;
beta  = 1.0;
gamma = 0.5;
cost  = 0.2;
```

Here, the discount factor, growth function parameters, demand function parameter, and unit cost of harvest are specified, respectively.

**Step 3** Specify basis functions and collocation nodes:

```
n = 8;
smin = 6;
smax = 9;
fspace = fundefn('cheb',n,smin,smax);
snodes = funnode(fspace);
```

Here, the first eight Chebychev polynomials on the interval $[6, 9]$ and the associated standard Chebychev nodes are selected to serve as basis functions and collocation nodes.

**Step 4** Pack model structure:

```
model.func = 'mfdp08';
model.discount = delta;
model.params = {alpha beta gamma cost};
```

Here, `model` is a structured variable whose fields contain the elements of the dynamic decision model. The first field contains the name of the model function file `'mfdp08'`; the remaining fields contain the discount factor `delta` and model function parameters `params`, respectively.

**Step 5** Provide judicious guesses for values and actions at the collocation nodes:

```
sstar = (alpha^2-1/delta^2)/(2*beta);
xstar = sstar - (delta*alpha-1)/(delta*beta);
[vlq,xlq] = lqapprox(model,snodes,sstar,xstar);
```

Here, the steady-state state and action are computed analytically and passed to the Compecon library routine `lqapprox`, which returns the linear-quadratic approximation values and actions at the collocation nodes. The linear-quadratic approximation is used to initialize the collocation algorithm.

**Step 6** Solve the decision model:

```
[c,s,v,x,resid] = dpsolve(model,fspace,snodes,vlq,xlq);
```

The Compecon routine `dpsolve` accepts as input the model structure `model`, basis functions `fspace`, collocation nodes `snodes`, and initial guesses for the value function `vlq` and optimal actions `xlq`. It then solves the collocation equation, returning as output the basis coefficients `c`, and the optimal values `v`, optimal actions `x`, and Bellman equation residuals `resid` at a refined state grid `s`.

**Step 7** Perform postoptimality analysis. Figure 9.4a gives optimal harvest as a percent of resource stock computed using both Chebychev collocation and linear-quadratic approximation. The Chebychev collocation approximant is upward sloping and the linear-quadratic approximant is downward sloping. Figure 9.4b gives the shadow price of the resource stock computed using both Chebychev collocation and linear-quadratic approximation. Both approximants are downward sloping, but the Chebychev approximant has a steeper slope. Figure 9.4c gives the Bellman equation residual for the Chebychev approximant. The residual possesses zeros at the collocation nodes by design and exhibits very nearly equal oscillations between the nodes. In this example, an eight degree Chebychev approximation was sufficient to solve the Bellman equation to a relative residual error of order $2 \times 10^{-10}$.

Figure 9.3d gives the path followed by the resource stock level over a twenty year period, beginning from a level of 6. The path was computed the Compecon library routine `dpsimul`:

```
nyrs = 20;
sinit = smin;
[spath,xpath] = dpsimul(model,sinit,nyrs,s,x);
```

As seen in this figure, resource stock rises rapidly, effectively converging to its steady-state value of 4.5 within four years.

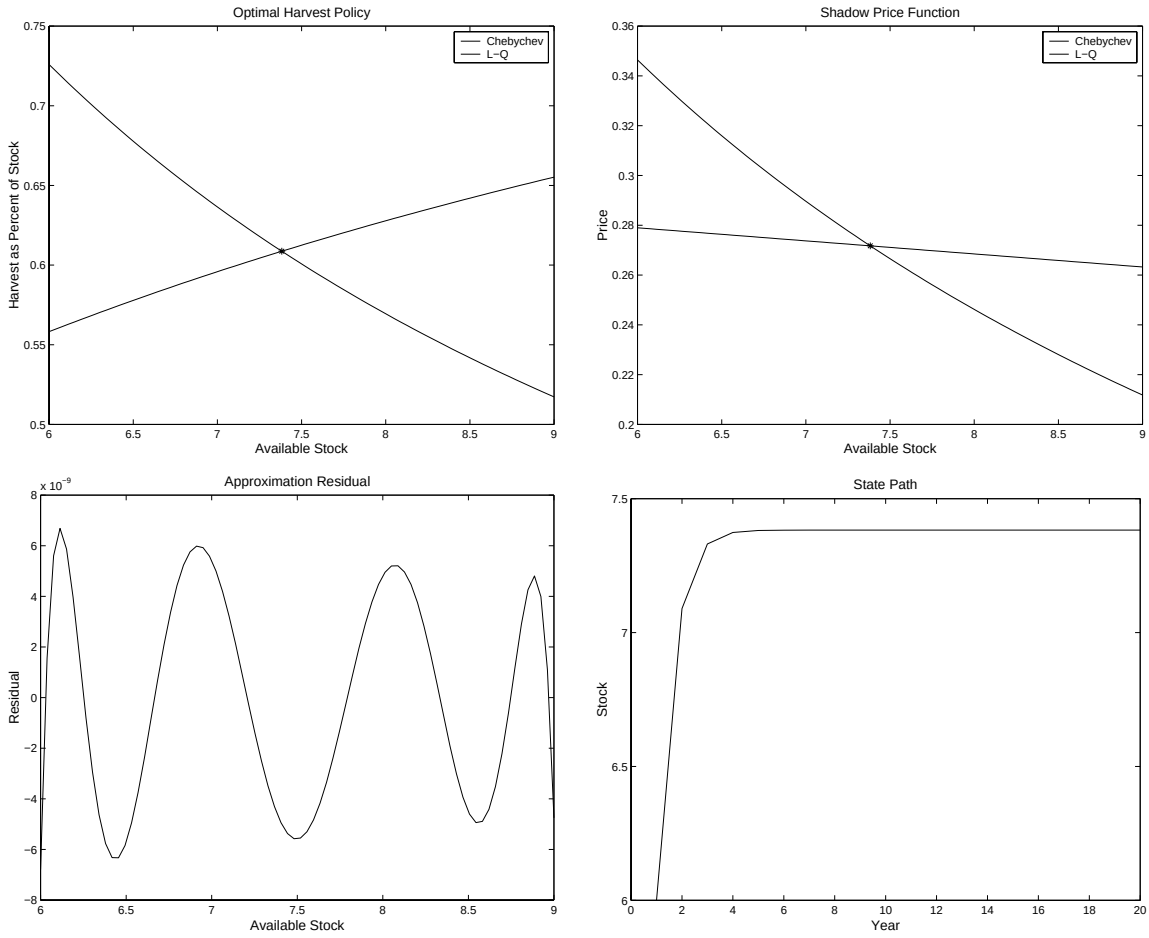


Figure 9.4: Solution to Public Renewable Resource Management Model

### 9.4.5 Private Nonrenewable Resource Management

Consider the private nonrenewable resource management model of Section 8.3.3 assuming a constant output price $\alpha$ and a cost of extraction $c(s,x) = x^2/(\beta + s)$ where $\alpha$ and $\beta$ are positive constants. The collocation method calls for the analyst to select $n$ basis functions $\phi_j$ and $n$ collocation nodes $s_i$, and form the value function approximant $V \approx \sum_{j=1}^{n} c_j \phi_j$ whose coefficients $c_j$ solve the collocation equation

$$\sum_{j=1}^{n} c_j \phi_j(s_i) = \max_{0 \le x \le s_i} \{\alpha\ x - x^2/(\beta + s) + \delta \sum_{j=1}^{n} c_j \phi_j(s_i - x)\}.$$

In practice, one may solve the private nonrenewable resource management model using Compecon library routines as follows:[8]

**Step 1** Code model function file:

```
function [out1,out2,out3] = mfdp09(flag,s,x,e,alpha,beta);
 switch flag
 case 'b'; % BOUND FUNCTION
   out1 = zeros(size(s));
   out2 = s;
 case 'f'; % REWARD FUNCTION
   out1 = alpha*x - (x.^2)./(beta+s);
   out2 = alpha - 2*x./(beta+s);
   out3 = -2./(beta+s);
 case 'g'; % STATE TRANSITION FUNCTION
   out1 =  s-x;
   out2 = -ones(size(s));
   out3 = zeros(size(s));
 end
```

The model function file returns the values and derivatives of the bound, reward, and transition functions at arbitrary vectors of states `s` and actions `x`. Passing the flag `'b'` returns the lower and upper bounds on the action `xl` and `xu`; passing the flag `'f'` returns the reward function value and its first and second derivatives with respect to the action, `f`, `fx`, and `fxx`; and passing the flag `'g'` returns the transition function value and its first and second derivatives with respect to the action, `g`, `gx`, and `gxx`.

**Step 2** Enter model parameters:

```
delta =  0.9;
alpha =  1.0;
beta  = 20.0;
```

Here, the discount factor, output price, and cost function parameter are specified, respectively.

**Step 3** Specify basis functions and collocation nodes:

[8] Functioning Matlab code for this example is contained in the Compecon library demonstration file `demdp09`.

```
n = 100;
smin = 0;
smax = 5;
fspace = fundefn('spli',n,smin,smax);
snodes = funnode(fspace);
```

Here, a 100 cubic spline basis on the interval $[0, 5]$ and the corresponding standard nodes are selected to serve as basis functions and collocation nodes.

**Step 4** Pack model structure:

```
model.func = 'mfdp09';
model.discount = delta;
model.params = {alpha beta};
```

Here, `model` is a structured variable whose fields contain the elements of the dynamic decision model. The first field contains the name of the model function file `'mfdp09'`; the remaining fields contain the discount factor `delta` and model function parameters `params`, respectively.

**Step 5** Provide judicious guesses for values and actions at the collocation nodes:

```
xinit = snodes;
vinit = zeros(size(snodes));
```

Here, since the model has no meaningful steady-state, the harvest is set equal to the stock level and the initial value function is set to zero to initialize the collocation algorithm.

**Step 6** Solve the decision model:

```
[c,s,v,x,resid] = dpsolve(model,fspace,snodes,vinit,xinit);
```

The Compecon routine `dpsolve` accepts as input the model structure `model`, basis functions `fspace`, collocation nodes `snodes`, and initial guesses for the value function `vinit` and optimal actions `xinit`. It then solves the collocation equation, returning as output the basis coefficients `c`, and the optimal values `v`, optimal actions `x`, and Bellman equation residuals `resid` at a refined state grid `s`.

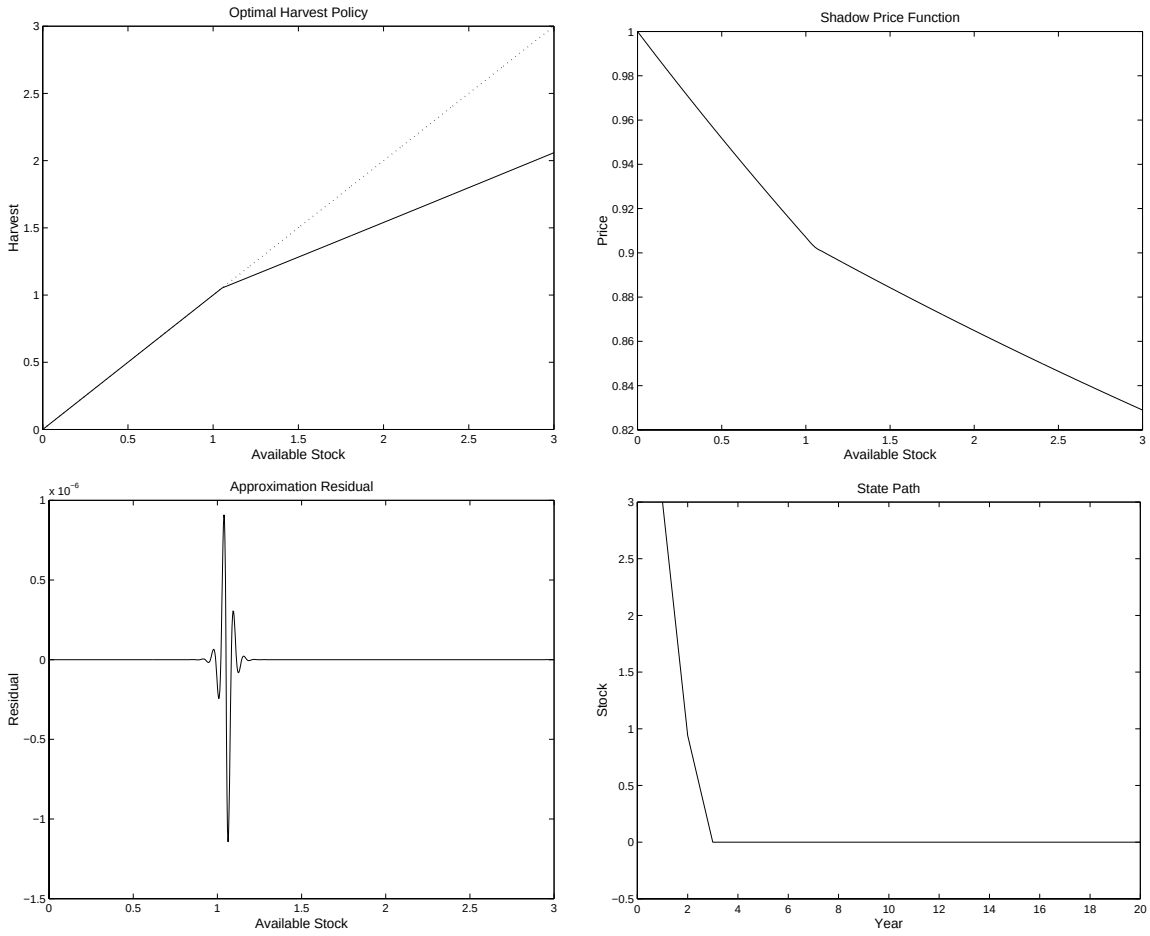


Figure 9.5: Solution to Private Nonrenewable Resource Management Model

**Step 7** Perform postoptimality analysis. Figure 9.5a gives optimal harvest policy and the $45^o$ line. A salient feature of the optimal policy is that, for stock levels roughly below 1, the upper bound on the extraction level is binding and the optimal policy is to extract all remaining stock. Figure 9.5b gives the shadow price of the resource stock. As seen in this figure, the shadow price exhibits an apparently discontinuous derivative or kink at the point at which the upper bound becomes binding. Figure 9.5c gives the Bellman equation residual for the cubic spline approximant. The residual exhibits a strong disturbance near the kink point, which is typical of spline and Chebychev polynomial approximations in the presence of kinks. Still, the residual produced with 100 cubic spline basis functions is small in the vicinity of the kink, on the order of $1 \times 10^{-6}$, and is several orders of magnitude smaller at stock levels

further removed from the kink. As seen in Figure 9.3d, if the initial stock level is 3, it will be optimal to extract the entire stock in exactly two years.

### 9.4.6 Optimal Monetary Policy

Consider the optimal monetary policy model of Section 8.3.5 assuming a loss function

$$L(s) = \frac{1}{2}(s - s^*)^\top \Omega (s - s^*)$$

and a state transition function

$$g(s, x, \epsilon) = \alpha + \beta s + \gamma x + \epsilon$$

where $\alpha$ and $\gamma$ are $2 \times 1$ constant vectors, $\beta$ is a $2 \times 2$ constant matrix, and $\epsilon$ is a $2 \times 1$ random with i.i.d. bivariate normal$(0, \Sigma)$ shock $\epsilon$. The collocation method calls for the analyst to select $n$ basis functions $\phi_j$ and $n$ collocation nodes $s_i$, and form the value function approximant $V \approx \sum_{j=1}^{n} c_j \phi_j$ whose coefficients $c_j$ solve the collocation equation

$$\sum_{j=1}^{n} c_j \phi_j(s_i) = \max_{0 \le x} \{-L(s_i) + \delta \sum_{k=1}^{m} \sum_{j=1}^{n} w_k c_j \phi_j(\alpha + \beta s + \gamma x + \epsilon_k)\}$$

where $\epsilon_k$ and $w_k$ represent Gaussian quadrature nodes and weights for the normal shock.

This example differs from the preceding continuous choice examples in that the state space is 2-dimensional. In practice, one may solve the optimal monetary policy model using Compecon library routines as follows:[9]

**Step 1** Code model function file:

```
function [out1,out2,out3] = mfdp11(flag,s,x,e,lambda,starget,a,b,c);
[n ds]  = size(s);
switch flag
case 'b'; % BOUND FUNCTION
  out1  = zeros(n,1);
  out2  = inf*ones(n,1);
case 'f'; % REWARD FUNCTION
  starget = starget(ones(n,1),:);
  out1 = -0.5*((s-starget).^2)*lambda';
```

[9]Functioning Matlab code for this example is contained in the Compecon library demonstration file `demdp11`.

```
  out2 = zeros(n,1);
  out3 = zeros(n,1);
case 'g'; % STATE TRANSITION FUNCTION
  out1 = a(ones(n,1),:) + s*b' + x*c + e;
  out2 = c(ones(n,1),:);
  out3 = zeros(n,ds);
end
```

The model function file returns the values and derivatives of the bound, reward, and transition functions at arbitrary vectors of states `s`, actions `x`, and shocks `e`. Passing the flag `'b'` returns the lower and upper bounds on the action `xl` and `xu`; passing the flag `'f'` returns the reward function value and its first and second derivatives with respect to the action, `f`, `fx`, and `fxx`; and passing the flag `'g'` returns the transition function value and its first and second derivatives with respect to the action, `g`, `gx`, and `gxx`.

**Step 2** Enter model parameters:

```
delta = 0.9;
a = [0.9 0.4];
b = [0.8 0.5; 0.2 0.6];
c = [-0.8 0.0];
lambda = [0.3 1];
starget = [0 1];
cov = 0.04*eye(2);
```

Here, the discount factor, transition function parameters, loss function preference weights, state targets, and shock covariance matrix are specified, respectively.

**Step 3** Discretize shock:

```
m   = [3 3];
mu  = [0 0];
[e,w] = qnwnorm(m,mu,cov);
```

Here, a 9-node discretization of the bivariate normal production shock is constructed by forming the Cartesian product of the three standard univariate nodes in each dimension.

**Step 4** Specify basis functions and collocation nodes:

```
n = [10 10];
smin = [-15 -10];
smax = [15 10];
fspace =   fundefn('spli',n,smin,smax); scoord = funnode(fspace);
snodes = gridmake(scoord);
```

Here, a 100-function bivariate Chebychev polynomial basis on the square $\{(s_1, s_2) | -15 \leq s_1 \leq 15, -10 \leq s_2 \leq 10\}$ is constructed by forming the tensor products of the first ten univariate Chebychev polynomials along each dimension; also, a 100-node collocation grid within the square is constructed by forming the Cartesian product of the ten standard Chebychev nodes along each dimension. Note that `snodes` will be a 100 by 2 matrix.

**Step 5** Pack model structure:

```
model.func = 'mfdp11';
model.discount = delta;
model.e = e;
model.w = w;
model.params = {lambda starget a b c};
```

Here, `model` is a structured variable whose fields contain the elements of the dynamic decision model. The first field contains the name of the model function file `'mfdp11'`; the remaining fields contain the discount factor `delta`, shock values `e`, shock probabilities `w`, and model function parameters `params`, respectively.

**Step 6** Provide judicious guesses for values and actions at the collocation nodes:

```
estar = [0 0];
sstar = starget;
xstar = (sstar(1)-a(1)-b(1,:)*sstar')/c(1);
[vlq,xlq] = lqapprox(model,snodes,sstar,xstar,estar);
```

Here, the the certainty-equivalent steady-state shock, state, and action are computed analytically and passed to the Compecon library routine `lqapprox`, which returns the linear-quadratic approximation values and actions at the collocation nodes. The linear-quadratic approximation is used to initialize the collocation algorithm.

**Step 7** Solve the decision model:

```
[c,s,v,x,resid] = dpsolve(model,fspace,snodes,vlq,xlq);
```

The Compecon routine `dpsolve` accepts as input the model structure `model`, basis functions `fspace`, collocation nodes `snodes`, and initial guesses for the value function `vlq` and optimal actions `xlq`. It then solves the collocation equation, returning as output the basis coefficients `c`, and the optimal values `v`, optimal actions `x`, and Bellman equation residuals `resid` at a refined state grid `s`.

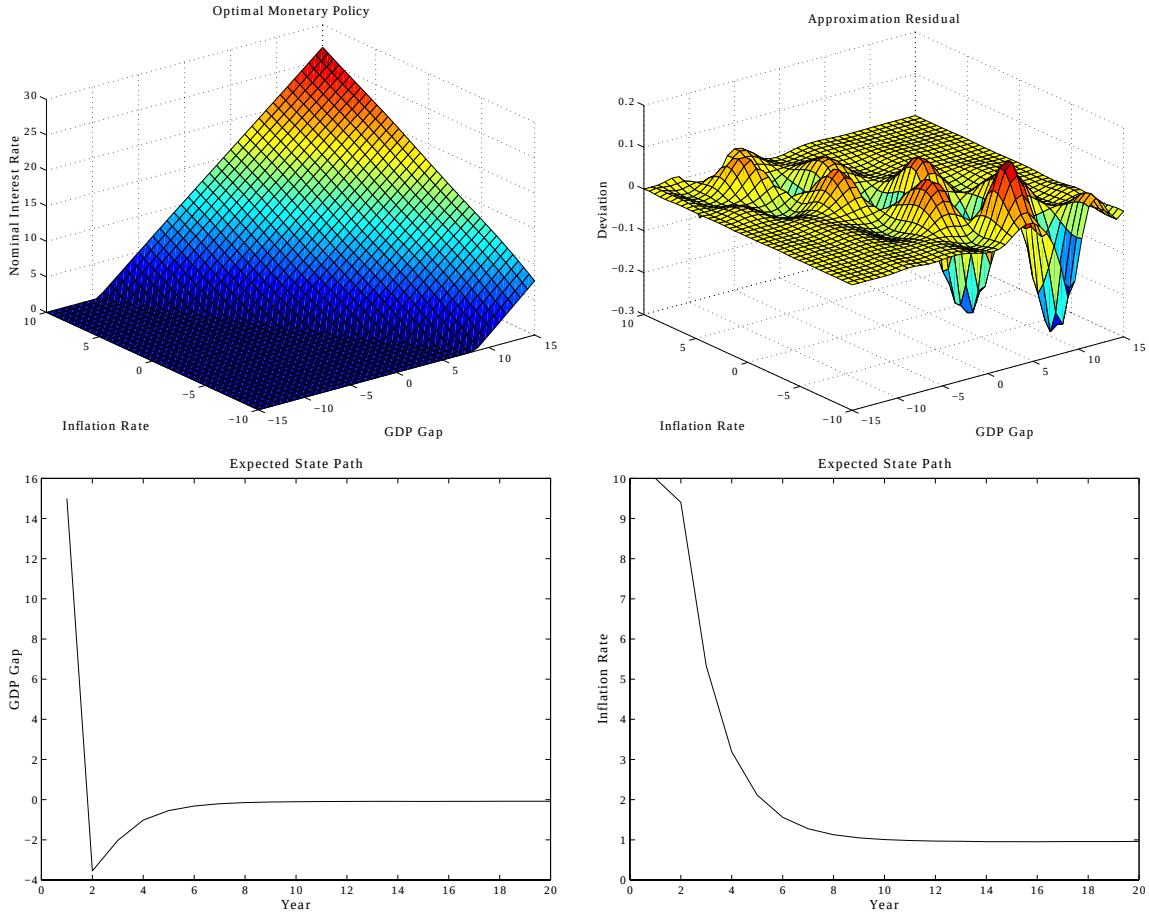


Figure 9.6: Solution to Optimal Monetary Policy Model

**Step 8** Perform postoptimality analysis. Figure 9.6a gives optimal nominal interest rate as a function of the underlying inflation rate and GDP gap. A salient feature of

the solution is that the nonnegativity constraint on the nominal interest rate is binding for low GDP gaps and inflation rates. Although the model possesses a quadratic objective and a linear state transition function, the exact solution cannot be derived via linear-quadratic approximation due to the binding constraint. Figure 9.6b gives the Bellman equation residual for the Chebychev approximant. The residual exhibits discernable turbulence along the boundary at which the nonnegativity constraint becomes binding, but is relatively small elsewhere. The residual can be reduced, but only at the expense of additional nodes along each direction. Unfortunately, doubling the basis functions and nodes in each direction quadruples the necessary computational effort due to the product rule construction of the bivariate basis and collocation grid.

The Compecon library routine `dpsimul` was used to simulate the model 5000 times over a twenty year period starting from an initial GDP gap of 10% and inflation rate of 15%. Figure 9.6c indicates that the expected GDP gap will initially drop dramatically, overshooting its target of zero, but over time converges asymptotically to its target. Figure 9.6d, on the other hand, indicates that the expected inflation rate will drop monotonically steadily, eventually converging to its target of 1.

### 9.4.7 Production-Adjustment Model

Consider the production-adjustment model of Section 8.3.6 assuming a linear cost of production function $c(q) = \kappa\, q$, a stochastic constant elasticity inverse demand curve $\eta q^{-\beta}$, a quadratic adjustment cost $a(q-l) = 0.5\alpha(q-l)^2$, and an i.i.d. lognormal$(0, \sigma^2)$ demand shock $\epsilon$. In practice, one may solve the production-adjustment model using Compecon library routines as follows:[10]

**Step 1** Code model function file:

```
function [out1,out2,out3] = mfdp12(flag,s,x,e,alpha,beta,kappa);
 n = size(s,1);
 l = s(:,1);
 d = s(:,2);
 q = x;
 switch flag
 case 'b'; % BOUND FUNCTION
   out1 = zeros(n,1);
   out2 = inf*ones(n,1);
 case 'f'; % REWARD FUNCTION
```

[10] Functioning Matlab code for this example is contained in the Compecon library demonstration file `demdp12`.

```
  out1 = d.*q.^(1-beta) - kappa*q - 0.5*alpha*((q-l).^2);
  out2 = (1-beta)*d.*q.^(-beta) - kappa - alpha*(q-l);
  out3 = -beta*(1-beta)*d.*q.^(-beta-1) - alpha;
case 'g'; % STATE TRANSITION FUNCTION
  out1 = [q e];
  out2 = [ones(n,1) zeros(n,1)];
  out3 = zeros(n,2);
end
```

The model function file returns the values and derivatives of the bound, reward, and transition functions at arbitrary vectors of states `s`, actions `x`, and shocks `e`. Passing the flag `'b'` returns the lower and upper bounds on the action `xl` and `xu`; passing the flag `'f'` returns the reward function value and its first and second derivatives with respect to the action, `f`, `fx`, and `fxx`; and passing the flag `'g'` returns the transition function value and its first and second derivatives with respect to the action, `g`, `gx`, and `gxx`.

**Step 2** Enter model parameters:

```
delta = 0.9;
beta  = 0.5;
kappa = 0.5;
alpha = 0.5;
sigma = 0.4;
```

Here, the discount factor, demand elasticity, unit production cost, marginal production cost, and demand shock volatility are specified, respectively.

**Step 3** Discretize shock:

```
m = 3;
[e,w] = qnwlogn(m,0,sigma^2);
```

Here, the lognormal demand shock is discretized using a three node Gaussian quadrature scheme.

**Step 4** Specify basis functions and collocation nodes:

```
n = [10 15];
smin = [xstar-1.0 e(1)];
smax = [xstar+3.0 e(m)];
```

```
fspace = fundefn('cheb',n,smin,smax);
scoord = funnode(fspace);
snodes = gridmake(scoord);
```

Here, a 150-function bivariate Chebychev polynomial basis is constructed by forming the tensor products of the first ten and first fifteen univariate Chebychev polynomials along first and second state dimension, respectively; also, a 150-node collocation grid is constructed by forming the Cartesian product of the ten and fifteen standard Chebychev nodes along each dimension. Note that `snodes` will be a 150 by 2 matrix.

**Step 5** Pack model structure:

```
model.func = 'mfdp12';
model.discount = delta;
model.e = e;
model.w = w;
model.params = {alpha beta kappa};
```

Here, `model` is a structured variable whose fields contain the elements of the dynamic decision model. The first field contains the name of the model function file `'mfdp12'`; the remaining fields contain the discount factor `delta`, shock values `e`, shock probabilities `w`, and model function parameters `params`, respectively.

**Step 6** Provide judicious guesses for values and actions at the collocation nodes:

```
estar = 1;
xstar = ((1-beta)/kappa)^(1/beta);
sstar = [xstar 1];
[vlq,xlq] = lqapprox(model,snodes,sstar,xstar,estar);
```

Here, the the certainty-equivalent steady-state shock, action, and state are computed analytically and passed to the Compecon library routine `lqapprox`, which returns the linear-quadratic approximation values and actions at the collocation nodes. The linear-quadratic approximation is used to initialize the collocation algorithm.

**Step 7** Solve the decision model:

```
[c,s,v,x,resid] = dpsolve(model,fspace,snodes,vlq,xlq);
```

The Compecon routine `dpsolve` accepts as input the model structure `model`, basis functions `fspace`, collocation nodes `snodes`, and initial guesses for the value function `vlq` and optimal actions `xlq`. It then solves the collocation equation, returning as output the basis coefficients `c`, and the optimal values `v`, optimal actions `x`, and Bellman equation residuals `resid` at a refined state grid `s`.

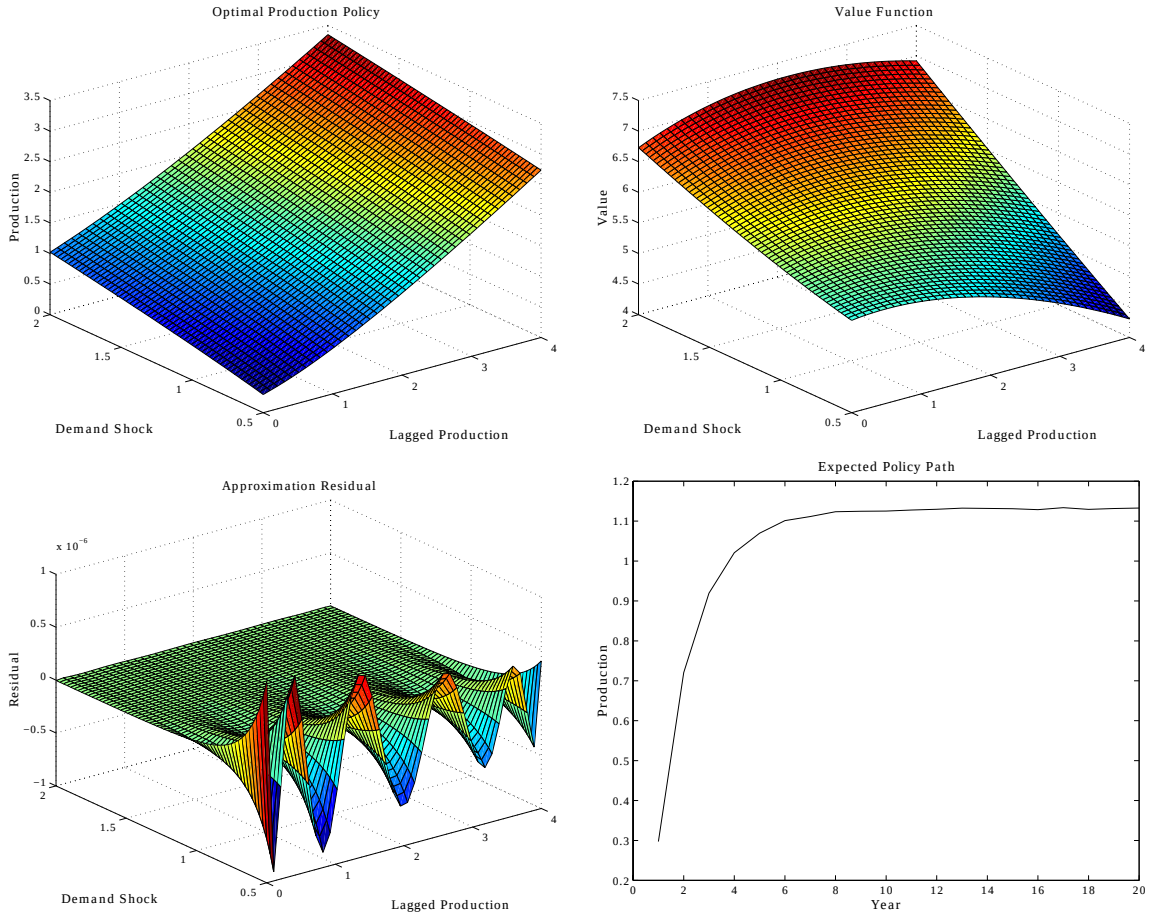


Figure 9.7: Solution to Production-Adjustment Model

**Step 8** Perform postoptimality analysis. Figure 9.7a shows that optimal production as a monotonically increasing function of the demand shock and the preceding period's production. Figure 9.7b gives the value of the firm as a function of the demand shock and the preceding period's production. Value is an increasing function of the demand shock, but a concave function of lagged production. Figure 9.7c gives the Bellman equation residual for the approximant. The residual exhibits discernable turbulence for low values of the demand shock. However, the maximum residuals are on the order of $1 \times 10^{-7}$ times the value of the firm. The Compecon library routine `dpsimul` was used to simulate the model 5000 times over a twenty year period starting from an production of 0.3. Figure 9.7d indicates that production can be expected to adjust gradually toward an steady-state mean value of approximately 1.1.

## 9.5 Dynamic Game Methods

Recall from Section 8.5 that the Markov perfect equilibrium of an $m$-agent infinite-horizon dynamic game is characterized by a set of $m$ simultaneous Bellman equations

$$V_p(s) = \max_{x_p \in X_p(s)} \left\{ f_p(s, x_p, x^*_{-p}(s)) + \delta E_\epsilon V_p(g(s, x_p, x_{-p}(s), \epsilon)) \right\},$$

$p = 1, 2, \ldots, m$, whose unknowns are the $m$ value functions $V_p(\cdot)$ and the associated optimal policies $x^*_p(\cdot)$, all of which are defined on the state space $S$. For the sake of discussion, assume that the state space is a bounded interval of Euclidian space, $S = [s_{min}, s_{max}]$, and that each agent $p$'s actions are constrained to an interval on the real line, $X_p(s) = [a_p(s), b_p(s)]$. Further assume that the reward functions $f_p$ and state transition function $g$ are twice continuously differentiable functions of their arguments.[11]

To compute an approximate solution to the system of Bellman functional equations via collocation, one employs the following strategy: First, write the value function approximants as linear combinations of known basis functions $\phi_1, \phi_2, \ldots, \phi_n$ whose coefficients $c_{p1}, c_{p2}, \ldots, c_{pn}$, are to be determined:

$$V_p(s) \approx \sum_{j=1}^{n} c_{pj} \phi_j(s).$$

Second, fix the $mn$ basis function coefficients $c_{p1}, c_{p2}, \ldots, c_{pn}$ by requiring the approximants to satisfy their respective Bellman equations, 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 $m$ Bellman functional equations with a system of $mn$ nonlinear equations in $mn$ unknowns. Specifically, to compute the value function approximants, or more precisely, to compute the $mn$ basis coefficients $c_{p1}, c_{p2}, \ldots, c_{pn}$ in their basis representations, one solves the equation system

$$\sum_j c_{pj} \phi_j(s_i) = \max_{x_{pi} \in X_p(s_i)} \{ f_p(s_i, x_{pi}, x_{-pi}) + \delta E_\epsilon \sum_{j=1}^{n} c_{pj} \phi_j(g(s_i, x_{pi}, x_{-pi}, \epsilon)) \}.$$

for $p = 1, 2, \ldots, m$ and $i = 1, 2, \ldots, n$, where $x_{pi}$ is action taken by agent $p$ and $x_{-pi}$ are the actions taken by his competitors when the state is $s_i$. The nonlinear equation system may be compactly expressed in vector form as a system of simultaneous collocation equations

$$\Phi c_p = v_p(c_p, x_{-p}).$$

[11] The presentation generalizes to models with dimensional individual action spaces, but at the expense of cumbersome additional notation that offers little additional insight.

Here, $c_p$ is the $n$ by 1 vector of basis coefficients of agent $p$'s value function approximant; $x_{-p}$ is the $(m-1)n$ vector of his competitors actions at the state nodes; $\Phi$, 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 $v_p$, agent $p$'s collocation function, is the function from $\Re^{mn}$ to $\Re^n$ whose typical $i^{th}$ element is

$$v_{pi}(c_p, x_{-p}) = \max_{x_{pi} \in X_p(s_i)} \{f_p(s_i, x_{pi}, x_{-pi}) + \delta E_\epsilon \sum_{j=1}^{n} c_{pj}\phi_j(g(s_i, x_{pi}, x_{-pi}, \epsilon))\}.$$

Agent $p$'s collocation function evaluated at a particular vector of basis coefficients $c_p$ yields an $n$ by 1 vector $v_p$ whose $i^{th}$ entry is the value obtained by solving the optimization problem embedded in Bellman's equation at the $i^{th}$ collocation node, replacing the value function appearing in the optimand with the approximant $\sum_j c_{pj}\phi_j$ and taking his competitor's current actions $x_{-p}$ as given.

Just as in case of single-agent dynamic optimization model, the simultaneous collocation equations of an $m$-agent game may be solved using standard nonlinear equation solution methods. However, one cannot solve the individual collocation equations independently because one agent's optimal action depends on the actions taken by others. Still, one can solve collocation equations for the $m$-agent using iterative strategies that are straightforward generalization of those used to solve single-agent dynamic optimization models. For example, one may write the collocation equation as a fixed-point problem $c_p = \Phi^{-1}v_p(c_p, x_{-p})$ and use function iteration, which employs the iterative update rule

$$c_p \leftarrow \Phi^{-1}v_p(c_p, x_{-p}).$$

In this implementation, the optimal actions of all agents are updated at each iteration with the evaluation of the collocation functions $v_p$, and passed as arguments to the collocation functions $v_p$ in the subsequent iteration.

Alternatively, one may write the collocation equation as a rootfinding problem $\Phi c_p - v_p(c_p, x_{-p}) = 0$ and solve for $c$ using a mixed Newton's method, which employs the iterative update rule

$$c_p \leftarrow c_p - [\Phi - v'(x)]^{-1}[\Phi c_p - v_p(c_p, x_{-p})].$$

Here, $v'(x)$ is the common $n$ by $n$ Jacobian of the collocation functions $v_p$ with respect to the basis coefficient $c_p$. The typical element of $v'(x)$ may be computed by applying the Envelope Theorem:

$$v'_{ij}(x) = \delta E_\epsilon \phi_j(g(s_i, x_i, \epsilon))$$

where $x_i$ is the vector of optimal actions taken by all $p$ players when the state is $s_i$.

As with the single-agent model, unless the m-agent game model is deterministic, one must compute expectations in a numerically practical way. Regardless of which quadrature scheme is selected, the continuous random variable $\epsilon$ in the state transition function is replaced with a discrete approximant, say, one that assumes values $\epsilon_1, \epsilon_2, \ldots, \epsilon_K$ with probabilities $w_1, w_2, \ldots, w_K$, respectively. In this instance, the collocation functions $v_p$ take the specific form

$$v_{pi}(c_p, x_{-p}) = \max_{x_{pi} \in X_p(s_i)} \{f_p(s_i, x_{pi}, x_{-pi}) + \delta \sum_{k=1}^{K} \sum_{j=1}^{n} w_k c_{pj} \phi_j(g(s_i, x_{pi}, x_{-pi}, \epsilon_k))\}.$$

and their Jacobian takes the form

$$v'_{ij}(x) = \delta \sum_{k=1}^{K} w_k \phi_j(g(s_i, x_i, \epsilon_k))$$

The practical steps that must be taken to implement the collocation method for dynamic games in a computer programming environment are similar to taken to solve single-agent models. The initial step is to specify the basis functions that will be used to express the value function approximants and the collocation nodes at which the Bellman equations will be required to hold exactly. This step may be executed using the Compecon library routines `fundefn`, `funnode`, and `funbas`, which are discussed in Chapter 6:

```
fspace = fundefn('cheb',n,smin,smax);
s      = funnode(fspace);
Phi    = funbas(fspace);
```

Here, it is presumed that the analyst has previously specified the lower and upper endpoints of the state interval, `smin` and `smax`, and the number of basis functions and collocation nodes `n`. After execution, `fspace` is a structured variable containing all the information needed to well-define the approximation space, `s` is the `n` by `1` vector of standard collocation nodes for the selected basis, and `Phi` is the associated `n` by `n` collocation matrix. In this specific example, the standard Chebychev polynomials basis functions and collocation nodes are used to form the approximant.

Next, a numerical routine must be coded to evaluate the collocation functions and their common derivative at an arbitrary set of basis coefficient vectors. A version of such a routine in which all basis coefficients and actions are efficiently stored in matrix formats with columns corresponding to different agents would have a calling sequence of the form

```
[v,x,vjac] = vmax(s,x,c).
```

Here, on input, `s` is an `n` by `1` vector of collocation nodes, `c` is an `n` by `m` matrix of basis coefficients, and `x` is an `n` by `m` matrix of current optimal actions. On output, `v` is an `n` by `m` matrix of optimal values at the collocation nodes, `x` is an `n` by `m` matrix of updated optimal actions at the nodes, and `vjac` is an `n` by `n` Jacobian of the collocation functions. The `m` columns of `v`, `x`, and `c` correspond to the `m` agents.

Given a `vmax` function coded as described, the joint collocation equations for the $m$-agent game can be solved by executing the same commands needed to solve the single-agent model. In particular, given the collocation nodes `s`, collocation matrix `Phi`, and collocation function routine `vmax`, and given initial guesses for the basis coefficient matrix `c` and optimal action matrix `x`, the collocation equation may be solved either by function iteration

```
for it=1:maxit
  cold = c;
  [v,x] = vmax(s,x,c);
  c = Phi\v;
  if norm(c-cold)<tol, break, end;
end
```

or by the Newton iteration

```
for it=1:maxit
  cold = c;
  [v,x,vjac] = vmax(s,x,c);
  c = cold - [Phi-vjac]\[Phi*c-v];
  if norm(c-cold)<tol, break, end;
end
```

The main challenge in implementing the collocation method for a dynamic game is coding the routine `vmax` that returns the collocation functions and their common derivative, which requires solving the optimization problems embedded in the $m$ Bellman equations at the collocation nodes. A simple routine that performs the optimizations by iteratively solving the associated Karush-Kuhn-Tucker complementarity conditions is as follows:

```
function [v,xnew,vjac] = vmax(s,xold,c) xnew = xold;
for p=1:m
  x = xold;
  [xl,xu] = bfunc(s,p);
  for it=1:maxit
    [f,fx,fxx] = ffunc(s,x,p);
    Ev=0; Evx=0; Evxx=0;
```

```
      for k=1:K
        [g,gx,gxx] = gfunc(s,x,e(k),p);
        vn     = funeval(c(:,p),fspace,g);
        vnder1 = funeval(c(:,p),fspace,g,1);
        vnder2 = funeval(c(:,p),fspace,g,2);
        Ev   = Ev   + w(k)*vn;
        Evx  = Evx  + w(k)*vnder1.*gx;
        Evxx = Evxx + w(k)*(vnder1.*gxx + vnder2.*gx.^2);
      end
      v    = f + delta*Ev;
      delx = -(fx+delta*Evx)./(fxx+delta*Evxx);
      delx = min(max(delx,xl-x),xu-x);
      x(:,p) = x(:,p) + delx;
      if norm(delx)<tol, break, end;
    end
    xnew(:,p) = x(:,p)
  end
  vjac = 0;
  for k=1:K
    g = gfunc(s,xnew,e(k));
    phinext = funbas(fspace,g);
    vjac = vjac + delta*w(k)*phinext;
  end
```

The $m$-agent game routine `vmax` differs from the similarly-named routine for single-agent models primarily in that $m$ Bellman optimands rather than one must be maximized. In this implementation, an outer loop over the agent index `p` is executed. For each agent, the collocation function is evaluated using the ancillary routines `bfunc`, `ffunc`, and `gfunc` that compute the bounds, rewards, and state transitions and additionally take the agent's index as an argument. In particular, the routine `bfunc` accepts the `n` by `1` vector of states `s` and the agent's index `p` and returns `n` by `1` vectors `xl` and `xu` of associated lower and upper bounds on the agent's actions. The routine `ffunc` accepts the `n` by `1` vector of states `s`, the `n` by `m` matrix of actions `x`, and the agent's index `p` and returns `n` by `1` vectors `f`, `fx`, and `fxx` of associated reward function values and derivatives with respect to the agent's actions. The routine `gfunc` accepts the `n` by `1` vectors of states `s`, the `n` by `m` matrix of actions `x`, the agent's index `p`, and a particular value of the shock `e` and returns `n` by `1` vectors `g`, `gx`, and `gxx` of associated state transition function values and derivatives with respect to the agent's actions.

Each outer agent loop in `vmax` begins by computing the lower and upper bounds `xl` and `xu` on agent `p`'s actions at the state nodes. The inner loop executes a series of Newton iterations that sequentially update agent `p`'s actions at the state nodes until the Karush-Kuhn-Tucker conditions of the optimization problem embedded in Bellman's equation are satisfied to a specified tolerance `tol`. With each iteration, the standard Newton step `delx` is computed and subsequently shortened, if necessary, to ensure that the updated action `x+delx` remains within the bounds `xl` and `xu`. The standard Newton step `delx` is the negative of the ratio of the second and third derivatives of agent `p`'s Bellman optimand with respect to agent `p`'s action, `fx+delta*Evx` and `fxx+delta*Evxx`.

In order to compute the expected value next period and its derivatives, another nested loop is executed over all `K` possible realizations of the discrete shock and probability weighted sums are formed (here, `e(k)` and `w(k)` are the $k^{th}$ shock and its probability). For each realization of the shock, the state next period `g` and its first and second derivatives with respect to agent `p`'s action, `gx` and `gxx`, are computed. The state next period is passed to the library routine `funeval`, which computes next period's value and its derivatives using the value function approximant that is identified with agent `p`'s coefficient vector `c(:,p)`. The Chain Rule is then used to compute the derivatives of the expected value.

Once convergence is achieved and the optimal value and action at the state nodes have been determined, the Jacobian `vjac` of the collocation function is computed. The Jacobian is a `n` by `n` matrix whose representative $ij^{th}$ entry is the discounted expectation of the $j^{th}$ basis evaluated at the following period's state, given the current state is the $i^{th}$ state node. To compute the Jacobian, a loop is executed over all `K` possible realizations of the discrete shock. For each realization of the shock, the state next period `g` is computed and passed to the Compecon library routine `funbas`, which in turn evaluates the basis functions at that state.

### 9.5.1 Capital-Production Game

Consider the capital-production game of Section 8.5.1 assuming that the market clearing prices $P_p = P_p(q_1, q_2)$ are given by

$$\begin{aligned} \log P_1 &= \log a_1 + e_{11} \log q_1 + e_{12} \log q_2, \\ \log P_2 &= \log a_2 + e_{21} \log q_1 + e_{22} \log q_2, \end{aligned}$$

that the cost of production is given by

$$C_p(q, k) = \gamma_p q_p^{\alpha} k_p^{\beta},$$

and new capital can be purchased at a constant price $\kappa$. The collocation method calls for the analyst to select $n$ basis functions $\phi_j$ and $n$ collocation nodes $(k_{1i}, k_{2i})$, and

form the value function approximants $V_p \approx \sum_{j=1}^{n} c_{pj}\phi_j$ whose coefficients $c_{pj}$ solve the collocation equation

$$\sum_{j=1}^{n} c_{pj}\phi_j(k_{1i}, k_{2i}) = \max_{q_p \geq 0, x_p \geq 0} \{P_p(q_1, q_2)q_p - C_p(q_p, k_p) - \kappa x_p + \delta E_\epsilon \sum_{j=1}^{n} c_j \phi_j(\hat{k}_1, \hat{k}_2)\}$$

where $\hat{k}_p = (1-\xi)k_p + x_p$. In practice, one may solve the capital-production game using Compecon library routines as follows:[12]

**Step 1** Code model function file:

```
function [out1,out2,out3] = mfgame01(flag,s,x,e,smax,eps,...
                                          gamma,A,beta,theta,alpha,cost,xi);
n = size(s,1);
ds = 2;
dx = 2;
switch flag
case 'b'; % BOUND FUNCTION
  out1 = zeros(n,dx);
  out2 = smax(ones(n,1),:) - (1-xi)*s;
case 'f1'; % REWARD FUNCTION
  Prof = Profit(s,n,alpha,beta,gamma,A,eps);
  c = cost(:,1);
  xx = x(:,1);
  f = Prof(:,1) - c*xx.^theta/theta;
  fx =  - c*xx.^(theta-1);
  fxx = - (theta - 1)*c*xx.^(theta - 2);
  out1 = zeros(n,1);
  out2 = zeros(n,dx);
  out3 = zeros(n,dx,dx);
  out1 = f;
  out2(:,1)= fx;
  out3(:,1,1) = fxx;
case 'f2'; % REWARD FUNCTION
  Prof = Profit(s,n,alpha,beta,gamma,A,eps);
  c = cost(:,2);
  xx = x(:,2);
  f = Prof(:,2) - c*xx.^theta/theta;
```

[12] Functioning Matlab code for this example is contained in the Compecon library demonstration file `demgame01`.

```
  fx =  - c*xx.^(theta-1);
  fxx = - (theta - 1)*c*xx.^(theta - 2);
  out1 = zeros(n,1);
  out2 = zeros(n,dx);
  out3 = zeros(n,dx,dx);
  out1 = f;
  out2(:,2)= fx;
  out3(:,2,2) = fxx;
case 'g'; % STATE TRANSITION FUNCTION
  g   = zeros(n,ds);
  gx  = zeros(n,ds,dx);
  gxx = zeros(n,ds,dx,dx);
  g = (1-xi)*s + x;
  gx(:,1,1) = ones(n,1);
  gx(:,2,2) = ones(n,1);
  out1=g; out2=gx; out3=gxx;
end
```

The model function file returns the values and derivatives of the bound, reward, and transition functions at arbitrary vectors of states `s`, actions `x`, and shocks `e`. Passing the flag `'b'` returns the lower and upper bounds on the action `xl` and `xu`; passing the flag `'f'` returns the reward function value and its first and second derivatives with respect to the action, `f`, `fx`, and `fxx`; and passing the flag `'g'` returns the transition function value and its first and second derivatives with respect to the action, `g`, `gx`, and `gxx`.

**Step 2** Enter model parameters:

```
delta   = 0.95;
A       = [1.5 1.5];
alpha   = [1.5 1.5];
beta    = -[0.75 0.75];
gamma   = [0.25 0.25];
eps     = -[0.5 0.2;0.2 0.5];
cost    = [1.5 1.5];
theta   =  2.5;
xi      = 0.07;
```

Here, the discount factor, demand parameters, cost elasticity of quantity, cost elasticity of capital, production cost parameters, demand elasticities, investment cost elasticity, and the depreciation rate are specified, respectively.

**Step 3** Specify basis functions and collocation nodes:

```
n = [8 8];
smin = [7 7];
smax = [10 10];
fspace = fundefn('cheb',n,smin,smax);
scoord = funnode(fspace);
s      = gridmake(scoord);
```

Here, an 64-function bivariate Chebychev polynomial basis is constructed by forming the tensor products of eight and eight Chebychev polynomials along the first and second state dimensions, respectively; also, an 64-node collocation grid is constructed by forming the Cartesian product of the eight and eight standard Chebychev nodes along the two state dimensions. Note that `snodes` will be a 64 by 2 matrix.

**Step 4** Pack model structure:

```
model.func = 'mfgame01';
model.discount = delta;
model.params={smax,eps,gamma,A,beta,theta,alpha,cost,xi};
```

Here, `model` is a structured variable whose fields contain the elements of the dynamic decision model. The first field contains the name of the model function file `'mfgame01'`; the remaining fields contain the discount factor `delta`, shock values `e`, shock probabilities `w`, and model function parameters `params`, respectively.

**Step 5** Provide judicious guesses for values and actions at the collocation nodes:

```
x = xi*s;
vinit = zeros(size(s,1),2);
vinit(:,1) = feval(model.func,'f1',s,x,[],model.params{:})/(1-delta);
vinit(:,2) = feval(model.func,'f2',s,x,[],model.params{:})/(1-delta);
```

Here, investment is initialized by setting it equal to depreciation and the value function is initialized by taking that level of investment and assuming the reward is constant over time.

**Step 6** Solve the decision model:

```
[c,s,v,x,resid] = gamesolve(model,fspace,snodes,v,x);
```

The Compecon routine `gamesolve` accepts as input the model structure `model`, basis functions `fspace`, collocation nodes `snodes`, and initial guesses for the value function `vlq` and optimal actions `xlq`. It then solves the collocation equation, returning as output the basis coefficients `c` and the optimal values `v`, optimal actions `x`, and Bellman equation residuals `resid` at a refined state grid `s`.

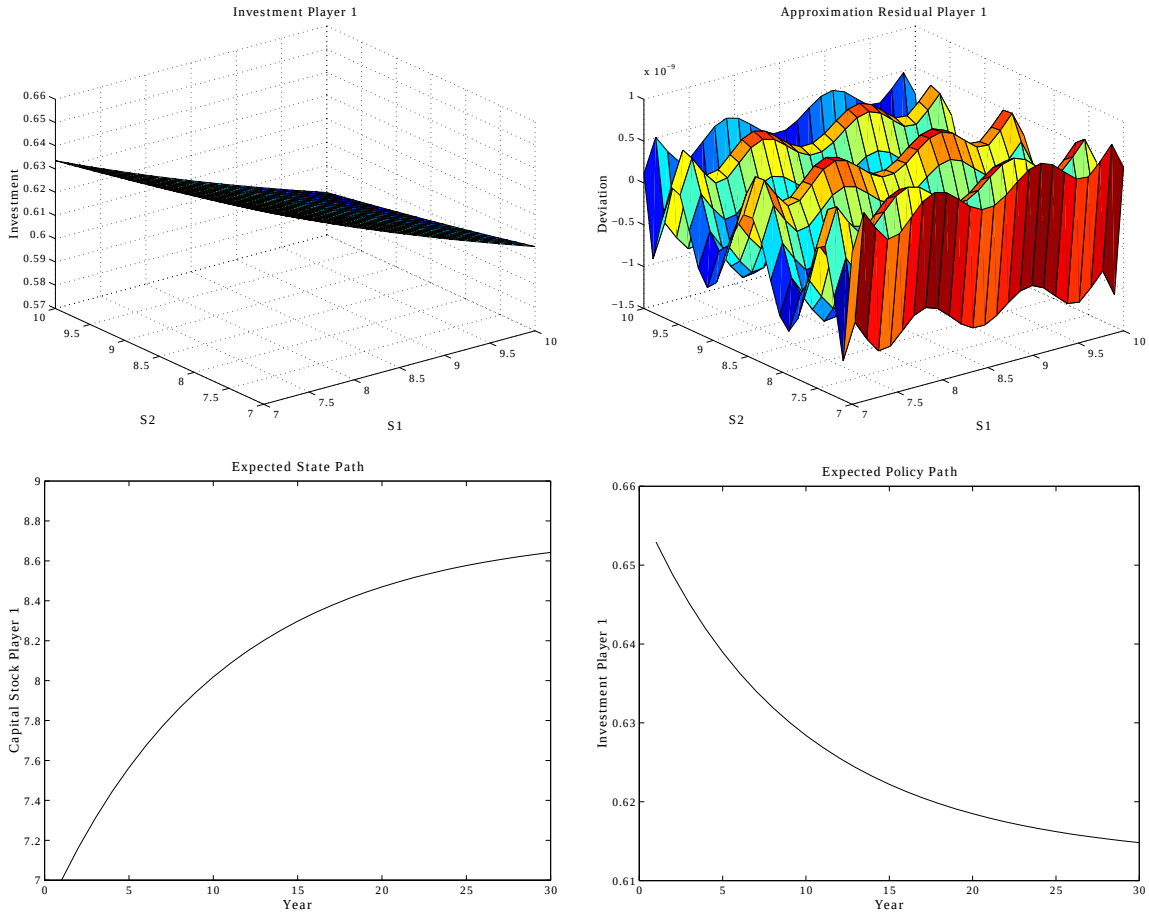


Figure 9.8: Solution to Capital-Production Game

## 9.5.2 Income Redistribution Game

Consider the capital-production game of Section 8.5.2 assuming that player $p$'s reward is $(s_p - x_p)^{1-\alpha_p}/(1-\alpha_p)$ and his wealth evolves according to $\gamma_p x_p + \epsilon_p x_p^{\beta_p}$, where the production shocks $\epsilon_p$ are i.i.d. lognormal$(0, \sigma^2)$. In practice, one may solve the income

redistribution game using Compecon library routines as follows:[13]

**Step 1** Code model function file:

```
function [out1,out2,out3] = mfgame02(flag,s,x,e,alpha,beta,gamma,share);
n = size(s,1);
ds = 2;
dx = 2;
switch flag
case 'b'; % BOUND FUNCTION
  xl = zeros(n,dx);
  xu = 0.99*s;
  out1=xl; out2=xu; out3=[];
case 'f1' % REWARD FUNCTION
  fx  = zeros(n,dx);
  fxx = zeros(n,dx,dx);
  f          = ((s(:,1)-x(:,1)).^(1-alpha(1)))/(1-alpha(1));
  fx(:,1)    = -(s(:,1)-x(:,1)).^(-alpha(1));
  fxx(:,1,1) = -alpha(1)*(s(:,1)-x(:,1)).^(-alpha(1)-1);
  out1=f; out2=fx; out3=fxx;
case 'f2' % REWARD FUNCTION
  fx  = zeros(n,dx);
  fxx = zeros(n,dx,dx);
  f          = ((s(:,2)-x(:,2)).^(1-alpha(2)))/(1-alpha(2));
  fx(:,2)    = -(s(:,2)-x(:,2)).^(-alpha(2));
  fxx(:,2,2) = -alpha(2)*(s(:,2)-x(:,2)).^(-alpha(2)-1);
  out1=f; out2=fx; out3=fxx;
case 'g'; % STATE TRANSITION FUNCTION
  g   = zeros(n,ds);
  gx  = zeros(n,ds,dx);
  gxx = zeros(n,ds,dx,dx);
  g1   = gamma(1)*x(:,1) + e(:,1).*x(:,1).^beta(1);
  g2   = gamma(2)*x(:,2) + e(:,2).*x(:,2).^beta(2);
  g(:,1)       = (1-share)*g1 + share*g2;
  gx(:,1,1)    = (1-share)*(gamma(1) + beta(1)*e(:,1).*x(:,1).^(beta(1)-1));
  gxx(:,1,1,1) = (1-share)*((beta(1)-1)*beta(1)*e(:,1).*x(:,1).^(beta(1)-2));
  g(:,2)       = (1-share)*g2 + share*g1;
  gx(:,2,2)    = (1-share)*(gamma(2) + beta(2)*e(:,2).*x(:,2).^(beta(2)-1));
```

[13]Functioning Matlab code for this example is contained in the Compecon library demonstration file `demgame02`.

```
    gxx(:,2,2,2) = (1-share)*((beta(2)-1)*beta(2)*e(:,2).*x(:,2).^(beta(2)-2));
    out1=g; out2=gx; out3=gxx;
end
```

The model function file returns the values and derivatives of the bound, reward, and transition functions at arbitrary vectors of states `s`, actions `x`, and shocks `e`. Passing the flag `'b'` returns the lower and upper bounds on the action `xl` and `xu`; passing the flag `'f'` returns the reward function value and its first and second derivatives with respect to the action, `f`, `fx`, and `fxx`; and passing the flag `'g'` returns the transition function value and its first and second derivatives with respect to the action, `g`, `gx`, and `gxx`.

**Step 2** Enter model parameters:

```
delta =  0.9;
alpha =  [0.2 0.2];
beta  =  [0.5 0.5];
gamma = [0.9 0.9];
sigma =  [0.1 0.1];
share = 0.05;
```

Here, the discount factor, utility function parameters, production elasticities, capital survival rates, production shock volatilities, and wealth share rate are specified, respectively.

**Step 3** Discretize shock:

```
m = 3;
[e,w] = qnwlogn(m,0,sigma^2);
```

Here, the lognormal demand shock is discretized using a three node Gaussian quadrature scheme.

**Step 4** Specify basis functions and collocation nodes:

```
n = [20 20];
smin = [3 3];
smax = [11 11];
fspace = fundefn('spli',n,smin,smax);
scoord = funnode(fspace);
s      = gridmake(scoord);
```

Here, an 400-function bivariate cubic spline basis is constructed by forming the tensor products of twenty and twenty basis functions along the first and second state dimensions, respectively; also, an 400-node collocation grid is constructed by forming the Cartesian product of the twenty and twenty standard cubic spline nodes along the two state dimensions. Note that `snodes` will be a 400 by 2 matrix.

**Step 5** Pack model structure:

```
model.func = 'mfgame02';
model.discount = delta;
model.e = e;
model.w = w;
model.params = {alpha beta gamma share};
```

Here, `model` is a structured variable whose fields contain the elements of the dynamic decision model. The first field contains the name of the model function file `'mfgame01'`; the remaining fields contain the discount factor `delta`, shock values `e`, shock probabilities `w`, and model function parameters `params`, respectively.

**Step 6** Provide judicious guesses for values and actions at the collocation nodes:

```
xstar = ((1-delta*gamma)./(delta*beta)).^(1./(beta-1));
sstar = gamma.*xstar + xstar.^beta;
xinit = 0.75*s;
vstar1 = feval(model.func,'f1',sstar,xstar,[],model.params{:});
vstar2 = feval(model.func,'f1',sstar,xstar,[],model.params{:});
vinit(:,1) = vstar1 + delta*(s(:,1)-sstar(1));
vinit(:,2) = vstar2 + delta*(s(:,2)-sstar(2));
```

Here, wealth and investment are initialized by setting them equal to their steady state values in the absence of any income sharing.

**Step 7** Solve the decision model:

```
[c,s,v,x,resid] = gamesolve(model,fspace,snodes,vinit,xinit);
```

The Compecon routine `gamesolve` accepts as input the model structure `model`, basis functions `fspace`, collocation nodes `snodes`, and initial guesses for the value function `vinit` and optimal actions `xinit`. It then solves the collocation equation, returning as output the basis coefficients `c` and the optimal values `v`, optimal actions `x`, and Bellman equation residuals `resid` at a refined state grid `s`.

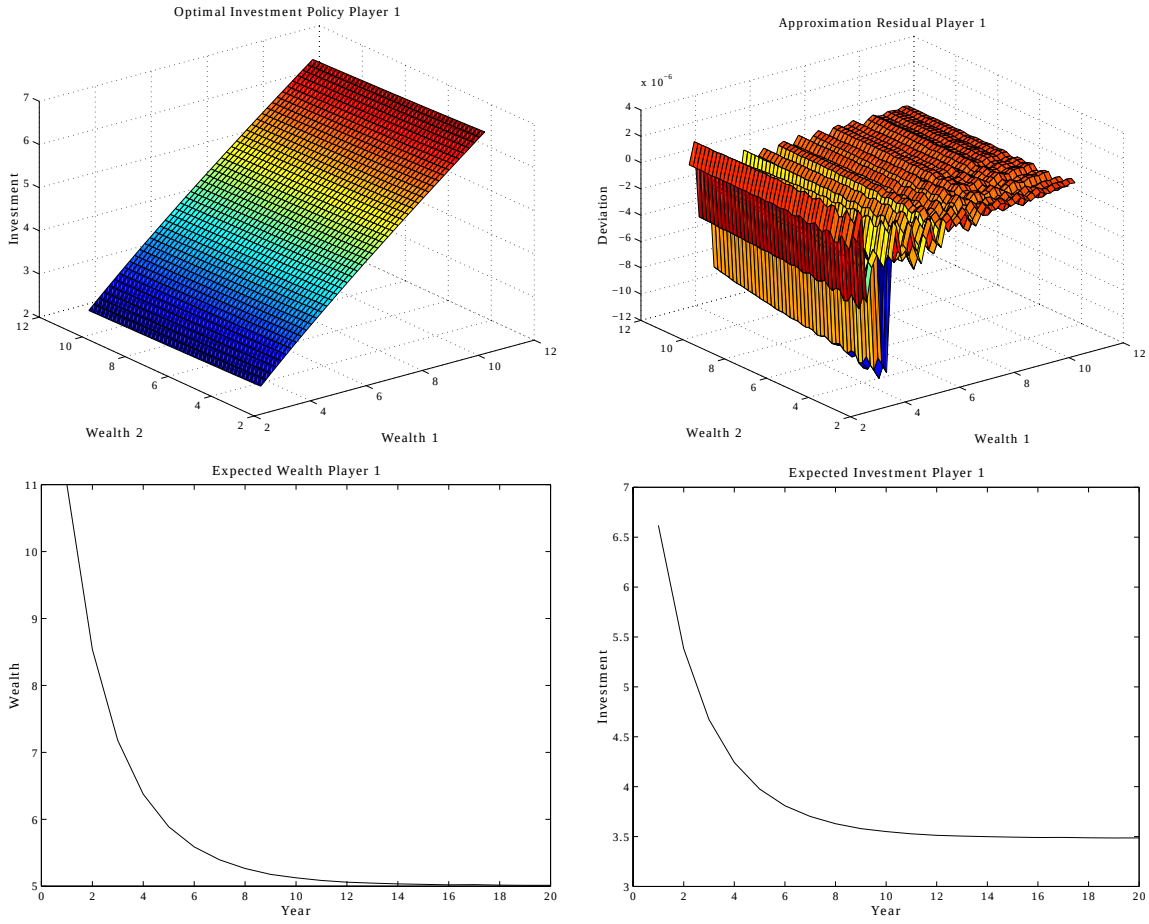


Figure 9.9: Solution to Income Redistribution Game

## 9.6 Rational Expectations Methods

Recall that the general formulation of the model involves solving for $x(s)$ a complementarity problem of the form

$$CP\Big(f(s_t, x_t, E_t h(s_{t+1}, x_{t+1})), a(s), b(s)\Big)$$

with $f : \Re^{n\times m\times p} \to \Re^m$ together with a state transition function

$$s_{t+1} = g(s_t, x_t, e_{t+1})$$

with $g : \Re^{n\times m\times n} \to \Re^n$.

This is an infinite dimensional problem but can be converted to a finite dimension one by using an approximating function for either $x^*(s)$ or $h(s, x^*(s))$. Although the former approach seems more natural, it can lead to difficulties, specially when the response function exhibits kinks due to binding constraints.

Response function approximation uses $x^*(s) \approx \phi(s)c$. Together with a $K$-valued discretization of the shock process $(e, w)$, we can write a residual function

$$r(c, s) = f\left(s, \phi(s)c, \sum_{k=1}^{K} w_k h(s'_k, \phi(s'_k)c)\right),$$

where $s'_k = g(s, \phi(s)c, e_k)$. Collocation can be used to determine $c$ by solving the complementarity problem $\min(\max(r(c, s_i), \phi(s_i)c - a(s_i)), \phi(s_i)c - b(s_i)) = 0$ at $N$ nodal values of $s$.

We have chosen, instead, to provide a general solver that works using $h(s, x^*(s)) \approx \phi(s)c$. For problems in which the response function is non-smooth due to boundary conditions, the non-smooth behavior will typically have less influence on the solution with this scheme because $h(s, x)$ will typically be smoother than $x$ itself.

For a given value of $c$, we determine $x$ by solving

$$CP\left(f\left(s, x, \sum_{k=1}^{K} w_k \phi(g(s, x, e_k))c\right), a, b\right).$$

To accomplish this it is useful to have the Jacobian with respect to $x$:[14]

$$\frac{df}{dx} = f_x + f_h\left[\sum_{k=1}^{K} w_k \phi'(g(s, x, e_k))c \; g_x(s, x, e_k)\right].$$

This requires that the partial derivatives $f_x$, $f_h$ and $g_x$ are available.

For a given set of $N$ $(s_i, x_i)$ pairs, $c$ $(N \times p)$ can be computed by solving the $N$-dimensional linear system

$$\phi(s_i)c = h(s_i, x^*(s_i)).$$

This leads naturally to a function iteration scheme, which is initialized by choosing a starting value for $c$. The coefficient matrix $c$ is updated iteratively by first solving the CP to obtain $x^{(j)}$ given $c^{(j)}$ and then performing the linear solve to obtain $c^{(j+1)}$ given $x^{(j)}$. The procedure continues until $||c^{(j+1)} - c^{(j)}||$ is less than some prescribed tolerance.[15]

[14] This is a shorthand notation. The sizes of these terms are $f_x$ $(m \times m)$, $f_h$ $(m \times p)$, $\phi'(g)c$ $(p \times n)$ and $g_x$ $(n \times m)$.

[15] This approach breaks down when $f_x = 0$ and $g_x = 0$ because $f$ given $c$ does not depend on $x$ and hence the equilibrium value of $x$ given $c$ is not defined.

We have implemented this approach in a MATLAB function `REMSOLVE`, the main features of which are now described (many of these steps parallel those used in solving dynamic programming problems; we will not discuss these in detail). First the user must specify the nature of the approximating function desired. For example (see discussion on p. 263):

```
fspace = fundefn('cheb',N,smin,smax);
s      = funnode(fspace);
Phi    = funbas(fspace);
```

Here, it is presumed that the analyst has previously specified the lower and upper endpoints of the state interval, `smin` and `smax`, and the number of basis functions and collocation nodes `N`.

Next, a numerical routine must be coded solve the arbitrage equation for arbitrary basis coefficients. Such a routine would have a calling sequence of the form

```
[f,x,h] = arbit(s,x,c).
```

Here, on input, `s` is an $N \times n$ matrix of collocation nodes, `x` is and $N \times m$ matrix of response levels and `c` is an $N \times p$ matrix of coefficients for approximating $h$. On output, `f` is an $N \times m$ matrix of values of $f(s, x^*(s), Eh)$ at the collocation nodes, `x` is an $N \times m$ matrix of values of $x^*(s)$ and `h` is an $N \times p$ matrix of values of $h(s, x^*(s))$. We describe this routine more fully below but for now suffice it to say that it finds $x^*(s_i)$ by solving $CP(f(s_i, x, Eh(s_i, x, c)), a(s_i), b(s_i))$ for each of the $N$ values of $s_i$. Thus $N$ separate $m$-dimensional CPs are solved rather than a single $Nm$-dimensional ones, an important computational advantage.

Given the collocation nodes `s`, collocation matrix `Phi`, and collocation function routine `arbit`, and given an initial guess for the basis coefficient vector `c`, the collocation equation may be solved by function iteration

```
for it=1:maxit
  cold = c;
  [f,x,h] = arbit(s,x,c);
  c = Phi\h;
  change = norm(c-cold,inf);
  if change<tol, break, end;
end
```

Here, `tol` and `maxit` are iteration control parameters set by the analyst, specifying the convergence tolerance and the maximum number of iterations.

The main challenge in implementing the collocation method is coding the routine `arbit` that solves for the equilibrium $x$ given $c$. First, however, we note that, to accomplish this, `arbit` must evaluate $f(s, x, Eh)$ and its derivative with respect to

$x$. Furthermore, in doing so, it must evaluate $Eh$, which is a function of $s$ and $x$. We demonstrate now how this is accomplished in a simplified setting ($m = 1$ and $p = 1$):[16]

```
function [f,fx]=equilibrium(s,x,c)
 K  = length(w);
 N=size(s,1);
 eh= 0;
 ehder = 0;
 for k=1:length(w)
  [g,gx]   = gfunc(s,x,e(k));
  hnext    = funeval(c,fspace,g);
  hnextder = funeval(c,fspace,g,1);
  eh       = eh    + w(k)*hnext;
  ehder    = ehder + w(k)*hnextder.*gx;
 end
 [f,fx,feh] = ffunc(s,x,eh);
 fx = fx + feh.*ehder;
```

This requires that the user has written a function to evaluate `gfunc` and `ffunc` along with the required derivatives. The specific way this is implemented is described below. The routine `ffunc` accepts an $N \times n$ matrix of state nodes `s`, an $N \times m$ matrix of response variable values `x`, and an $N \times p$ matrix of values of $Eh$. It returns an $N \times m$ matrix of $f$ values, an $N \times m \times m$ matrix of derivatives with respect to $x$ and an $N \times m \times p$ matrix of derivatives with respect to $Eh$.

The function for computing the equilibrium value of $x$ given $c$ can now be described.

```
function [f,x,h] = arbit(s,x,c)
for it=1:maxit
  xold = x;
  [f,fx]=equilibrium(s,x,c);
  [f,fx] = minmax(x,xl,xu,f,fx);
  deltax=-(fx\f);
  x = x + deltax;
  if norm(deltax(:))< tol, break, end;
end
h = hfunc(s,x);
```

[16]For clarity, the code omits several bookkeeping operations and programming tricks that accelerate execution. Operational versions of the code that efficiently handle arbitrary dimensional state and actions spaces are included with the Compecon library routine `remsolve`.

This requires that the user has written a function to evaluate `hfunc` and that the lower and upper bound functions on the response variables, `xl` and `xu`, have been defined (both are $N \times m$). The routine `hfunc` accepts an $N \times n$ matrix of state nodes `s` and an $N \times m$ matrix of response variable values `x`, and returns an $N \times p$ matrix of values of $h$.

Essentially `arbit` is a simple complementarity solver that uses Newton's method with no backstepping routine on the minmax transformation of the CP (see discussion in section 3.7 on page 48). A semi-smooth transformation of the CP can be used obtained by substituting the `smooth` function for `minmax` in the fifth line (this is a setable option in the toolbox version).

It is important to point out that the algorithm involves a double iteration. The inner iteration computes the equilibrium $x$ for fixed $c$ (done in `arbit`) via Newton's method. The outer iteration computes the equilibrium interpolating value of $c$ given a prior guess of its value, i.e., it uses function iteration to determine $c$.

### 9.6.1 Asset Pricing Model

In some models the approach just described is not needed. Consider the simple asset pricing model of Section 8.6.1 involving an exogenous state process for dividends, $d$, governed by

$$d_{t+1} = g(d_t, e_{t+1}),$$

and share price, $p$, as the response variable, governed by the equilibrium condition

$$U'(d)p(d) - \delta E_\epsilon\Big[U'\Big(g(d,\epsilon)\Big)\Big(p(g(d,\epsilon)) + g(d,\epsilon)\Big)\Big] = 0.$$

In the notation of the general model the expectation variable is

$$h(d,p) = U'(d)(p+d),$$

and equilibrium condition is

$$f(d,p,Eh) = U'(d)p - \delta Eh.$$

For this model it is perhaps easiest to seek an approximation of the response variable by using $p(d) \approx \phi(d)c$. The collocation method requires that for the analyst select $N$ basis functions $\phi_j$ and $N$ collocation nodes $s_i$, and solve for $c$ the linear function

$$U'(d)\phi(d)c - \delta E_\epsilon\Big[U'\Big(g(d,\epsilon)\Big)\Big(\phi(g(d,\epsilon))c + g(d,\epsilon)\Big)\Big] = 0$$

or, equivalently,

$$\Big(U'(d)\phi(d) - \delta E_\epsilon\Big[U'\Big(g(d,\epsilon)\Big)\phi(g(d,\epsilon))\Big]\Big)c = \delta E_\epsilon\Big[U'\Big(g(d,\epsilon)\Big)g(d,\epsilon)\Big]$$

To make this concrete, let $U(c) = (c^{1-\beta} - 1)/(1-\beta)$, so $U'(c) = c^{-\beta}$. Further, let

$$d_{t+1} = g(d_t, \epsilon_{t+1}) = \bar{d} + \gamma(d_t - \bar{d}) + \epsilon_{t+1}$$

where $\epsilon \sim i.i.d.\ N(0, \sigma^2)$. To compute the expectations we can use Gaussian quadrature nodes and weights for the normal shock ($\epsilon_k$ and $w_k$). The model can be solved by the following steps:[17]

**Step 1** Enter model parameters:

```
delta = 0.9;
dbar  = 1.0;
gamma = 0.5;
beta  = 0.4;
sigma = 0.1;
```

**Step 2** Discretize shock:

```
m = 3;
[e,w] = qnwnorm(m,0,sigma^2);
```

Here, the normal dividend shock is discretized using a three node Gaussian quadrature scheme.

**Step 3** Specify basis functions and collocation nodes:

```
n = 10;
dmin = dbar+min(e)/(1-gamma);
dmax = dbar+max(e)/(1-gamma);
fspace = fundefn('cheb',n,dmin,dmax);
dnode = funnode(fspace);
```

Here, the first ten Chebychev polynomials and the corresponding standard Chebychev nodes are selected to serve as basis functions and collocation nodes. The minimum and maximum values of $d$ are selected to ensure that $d_{t+1}$ never requires extrapolation: $min(d_{t+1}) = d_{min}$ when $d_t = d_{min}$ and $max(d_{t+1}) = d_{max}$ when $d_t = d_{max}$.

[17] Functioning Matlab code for this example is contained in the Compecon library demonstration file `demrem01`.

**Step 4** Solve the decision model:

```
LHS = diag(dnode.^(-beta))*funbas(fspace,ynode);
RHS = 0;
for k=1:m
 dnext = dbar + gamma*(dnode-dbar) + e(k);
 LHS   = LHS - delta*w(k)*diag(dnext.^(-beta))*funbas(fspace,dnext);
 RHS   = RHS + delta*w(k)*dnext.^(1-beta);
end
c = LHS\RHS;
```

The computed value of $c$ provides coefficients for the approximate pricing function, which is plotted along with the solution residuals in Figure 9.10.

**Step 5** Compute the response function and approximation residuals:

```
d = nodeunif(10*n,dmin,dmax);
p = funeval(c,fspace,d);
Eh=0;
for k=1:m
 dnext = dbar + gamma*(d-dbar) + e(k);
 h     = diag(dnext.^(-beta))*(funeval(c,fspace,dnext)+dnext);
 Eh    = Eh + delta*w(k)*h;
end
resid = d.^(-beta).*funeval(c,fspace,d)-Eh;
```

### 9.6.2 Competitive Storage

To solve the competitive storage model of section 8.6.2 we return to the use the general solver for rational expectations models, `REMSOLVE`. Recall that the equilibrium storage function is characterized by the functional complementarity condition

$$\delta E_y[P(x(s)+y-x(x(s)+y))]-P(s-x(s))-c=\mu(s)$$

$$x(s)\geq 0, \qquad \mu(s)\leq 0, \qquad x(s)>0 \Longrightarrow \mu(s)=0.$$

The carryin stocks level is the state variable and we treat the carryout stocks as the response variable. In the notation of the general model

$$h(s,x)=P(s-x),$$

$$g(s,x,y)=s+y-x$$

and

$$f(s,x,Eh)=\delta Eh-P(s-x)-c.$$

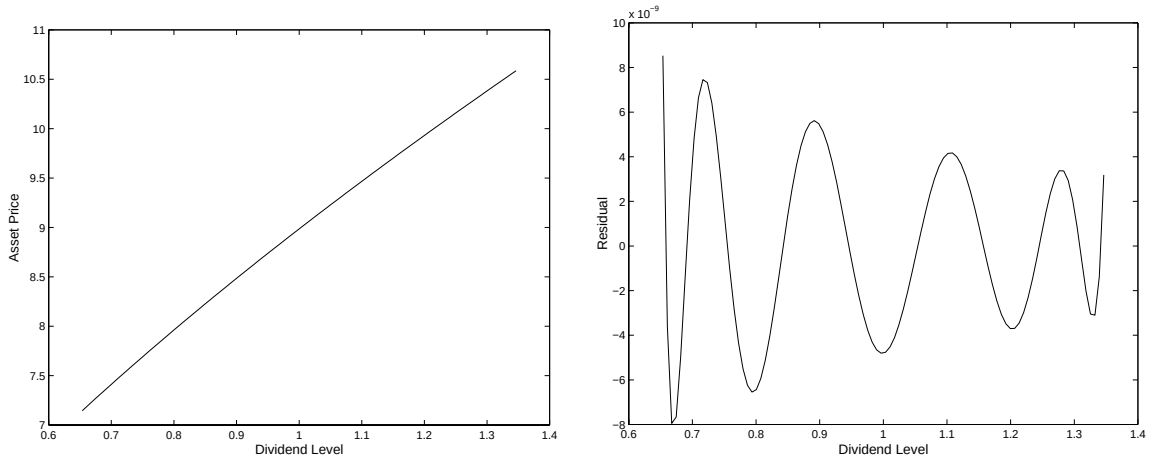


Figure 9.10: Solution to Asset Pricing Model

**Step 1** Code model function file:

```
function [out1,out2,out3]=prem02(flag,s,x,ep,e,delta,gamma,cost,xmax);
n=length(s);
switch flag
case 'b'; % BOUND FUNCTION
  out1 = zeros(n,1);
  out2 = max(s,xmax);
case 'f'; % EQUILIBRIUM FUNCTION
  out1 = delta*ep-(s-x).^(-gamma)-cost;
  out2 = -gamma*(s-x).^(-gamma-1);
  out3 = delta*ones(n,1);
case 'g'; % STATE TRANSITION FUNCTION
  out1 = x + e;
  out2 = ones(n,1);
case 'h'; % EXPECTATION FUNCTION
  out1 = (s-x).^(-gamma);
  out2 = gamma*(s-x).^(-gamma-1);
end
```

The model function file returns the values and (where appropriate) the derivatives of the bound, equilibrium, transition and expectation functions functions at arbitrary vectors of states `s`, actions `x`, next period's expectations `Eh` and shocks `e` (as well as the model specific additional parameters `delta`, `gamma`, `cost` and `xmax`). Passing the flag `'b'` returns the lower and upper bounds on the response variable `xl` and `xu`;

passing the flag `'f'` returns the equilibrium function value and its first derivatives with respect to the response and the expectation variables, `f`, `fx`, and `fh`; passing the flag `'g'` returns the transition function value and its first with respect to the response variable, `g` and `gx`; passing the flag `'h'` returns the expectation variable and its first derivative with respect to response variable, `h` and `hx`.

**Step 2** Enter model parameters:

```
delta = 0.9;
cost  = 0.1;
gamma = 2.0;
xmax =  0.9;
yvol  = 0.2;
```

Here, the discount factor, storage cost parameter, inverse demand elasticity, maximum storage level and yield volatility are specified.

**Step 3** Discretize shock:

```
m = 5;
[yshk,w] = qnwlogn(m,0,yvol^2);
```

Here, the lognormal production shock is discretized using a five node Gaussian quadrature scheme.

**Step 4** Pack model structure:

```
model.func = 'prem02';
model.discount = delta;
model.e = yshk;
model.w = w;
model.params = {delta,gamma,cost,xmax};
```

Here, `model` is a structure variable whose fields contain the elements of the model. The first field contains the name of the model function file `'prem02'`; the remaining fields contain the discount factor `delta`, shock values `e`, shock probabilities `w`, and model function parameters `params`, respectively.

**Step 5** Specify basis functions and collocation nodes:

```
n      = 10;
smin   = min(yshk);
smax   = max(yshk)+xmax;
fspace = fundefn('spli',n,smin,smax);
snodes = funnode(fspace);
```

Here, a degree 10 cubic spline with evenly spaced breakpoints and standard nodes are selected to serve as basis functions and collocation nodes. The domain of the state is taken to be the interval from minimum production level to the sum of the maximum production level and the maximum storage level. This ensures that next period's carryin stocks cannot lead to extrapolation beyond the approximation interval.

**Step 6** Provide a judicious guess for the response variable at the collocation nodes:

```
x = zeros(size(snodes));
```

Here, the carryout stocks are simply set to 0.

**Step 7** Solve equilibrium model:

```
[ch,sres,xres,hres,fres,resid] = remsolve(model,fspace,snodes,x);
```

The Compecon routine `remsolve` accepts as input the model structure `model`, basis function definition variable `fspace`, collocation nodes `snodes`, and the initial guess for the response variable `x`. It then solves the collocation equation, returning as output `c`, the basis coefficients for the expectation variable, a vector of the values of the state `sres`, together with values of the response variable `xres`, the expectation variable `hres`, the equilibrium function `fres` and the collocation residual `resid`, all evaluated at the values in `sres`.

**Step 8** Perform post-solution analysis. To check on the quality of the solution, both `fres` and `resid` should be examined. The former is used to demonstrate how well the equilibrium condition is satisfied when evaluated as

$$f(s, x(s;c), Eh(\phi(g(s, x(s;c), e))c);$$

the latter is used to demonstrate how well the approximating function matches the expectation variable:

$$resid = h(s, x(s;c)) - \phi(s)c.$$

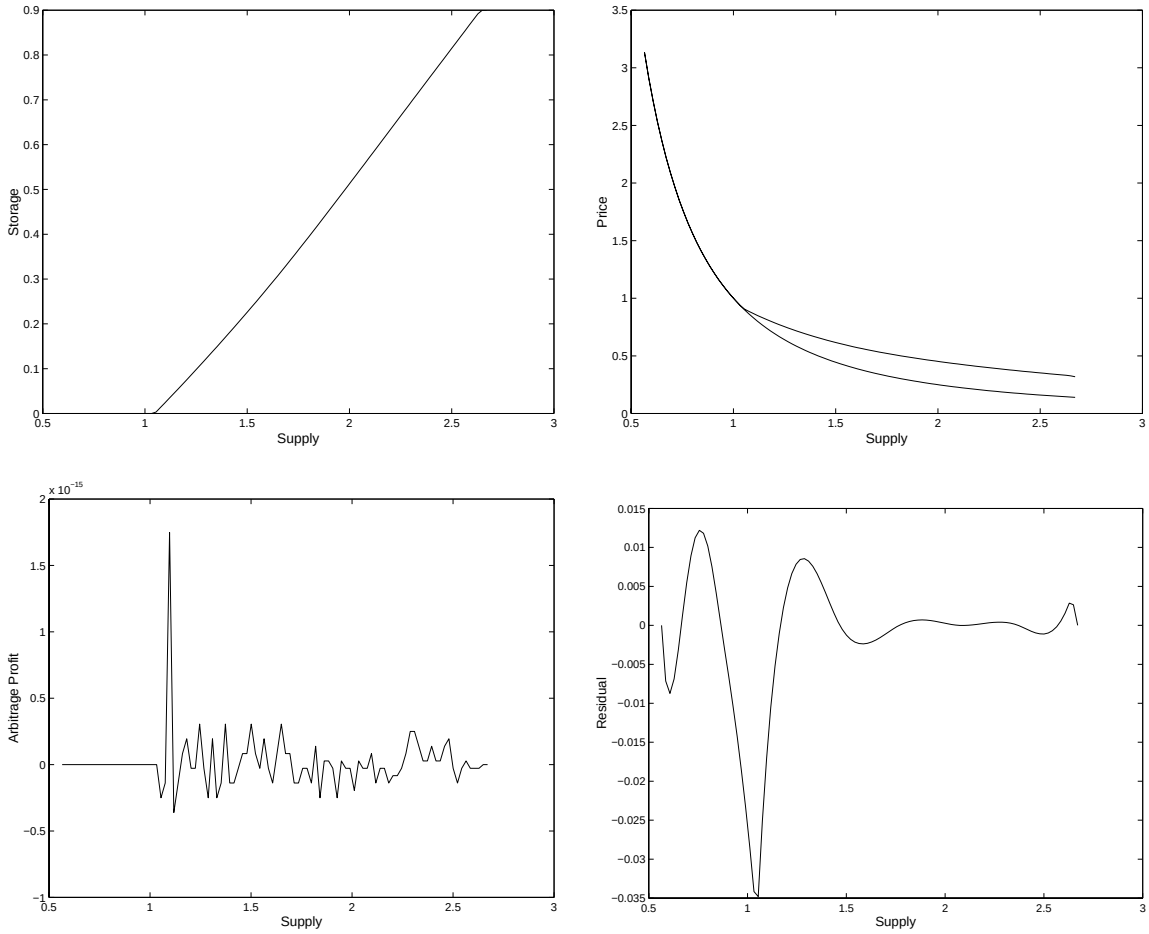


Figure 9.11: Solution to Commodity Storage Model

`resid` should be close to 0 for all $s \in [s_{min}, s_{max}]$; `fres` should be close to 0 for all values of $s$ such that $x(s)$ is on the interior of $[a(s), b(s)]$. Together these functions provide an indication of how accurately the solution has been approximated.

Figure 9.11a gives the equilibrium carryout stocks as a function of supply (carryin plus production). Figure 9.3b gives the equilibrium price function with stockholding (above) and without stockholding (below). Figure 9.11c shows that $f$ is satisfied to near machine accuracy. The collocation equation, however, shown in Figure 9.11d, indicates that the solution error is small but non-trivial (approximately 1%).

Although a higher degree approximate would provide a more accurate solution, Figures 9.11a and 9.11b exhibit the essential properties of the true rational expectations equilibrium solution. Specifically, when supply is low, price is high and storage

is zero; as supply rises, however, prices drop and stockholding becomes profitable. Both the response function (carryout stocks) and the expectation function (price) exhibit kinks at the point at which stockholding becomes profitable. It is precisely for this reason that we used a spline approximate. Furthermore, the price function is less drastically kinked and it therefore makes good sense to attempt to approximate it, rather than the stockholding function.

## 9.7 Comparison of Solution Methods

When applying the collocation method, the analyst faces a number of practical decisions. First, the analyst must choose the basis function and collocation nodes. Second, the analyst must chose an algorithm for solving the resulting nonlinear equation or complementarity problem. And third, the analyst must select an appropriate numerical quadrature technique for computing expectations. 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. If the basis functions and nodes are chosen wisely, the collocation method will often be numerically consistent, in the sense that the approximation error can be made arbitrarily small simply by increasing the number of basis functions, collocation nodes, and quadrature points.

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 dis-

cretization 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 9.1. 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.

Chebychev and cubic spline 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 9.6.2. 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.

| 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

**Exercises**

9.1. Rework the problem in the preceding chapter on optimal production with pollution externalities treating the state space as continuous, rather than discrete. Use a 10 degree Chebychev polynomial basis with nodes the Chebychev nodes for the interval $[2, 7]$.

(a) Compute the optimal shadow price and production policy using 10 degree Chebychev collocation.

(b) Graph the shadow price function obtained in (a) and the shadow price function obtained in problem 8.2 on the same figure.

(c) Graph the optimal production policy obtained in (a) and the optimal production policy obtained in problem 8.2 on the same figure.

(d) Plot pollution level through year 20, beginning with a pollution level of 7 at time 0.

(e) Plot production through year 20, beginning with a pollution level of 7 at time 0.

9.2. A farmer's corn yield in year $t$, in bushels per acre, is

$$y_t = 100 + 1.085(s_t + x_t) - 0.015(s_t + x_t)^2$$

where $s_t$ is soil fertilizer carryin and $x_t$ is fresh fertilizer applied topically at planting time, both measured in pounds per acre. The soil carryover dynamic is

$$s_{t+1} = 4.0 + 0.7s_t + 0.2x_t.$$

Develop a numerical dynamic program to maximize the discounted sum of profits over an infinite-horizon assuming that (i) the price of corn is \$2.00 per bushel; (ii) commercial fertilizer costs \$0.55 per pound; and (iii) the discount factor is 0.90.

(a) Derive the optimal fertilizer policy and shadow price function.

(b) Graph the optimal sequence of carryover levels assuming an initial stock of $s = 10$.

9.3. Consider an infinitely-lived put option with strike price $K$ on a financial asset whose log-price $p_t$ follows

$$p_{t+1} = \bar{p} - \gamma(p_t - \bar{p}) + \epsilon_{t+1}$$

where $\epsilon_t$ is i.i.d. normal with mean zero and standard deviation $\sigma$. Assuming $K = 1$, $\bar{p} = 0$, $\gamma = 0.5$, and $\sigma = 0.2$, price the option in terms of the log of the asset price over the range $[\log(0.8), \log(1.2)]$. What is the optimal exercise rule?

9.4. As a social planner, you wish to maximize the discounted sum of net social surplus from harvesting a renewable resource over an infinite horizon. Let $s_t$ denote the amount of resource available at the beginning of year $t$ and let $x_t$ denote the amount harvested. The harvest cost is $c(x_t) = kx_t$, the market clearing price is $p_t = x_t^{-\gamma}$, and the stock dynamic is $s_{t+1} = \alpha(s_t - x_t) - 0.5\beta(s_t - x_t)^2$. Assume $\gamma = 0.5$, $\alpha = 4$, $\beta = 1.0$, $k = 0.2$, and $\delta = 0.9$.

(a) Develop a computer program that will compute an approximate optimal policy using space discretization. Let $S = [1, 4]$ and $X = [1, 6]$, employing a 26 point discretization. Use the FORTRAN intrinsic function "nint" to find the state node closest to $g(s_i, x_j)$.

(b) Derive analytical expressions for the steady-state state $s^*$, action $x^*$, and shadow price $\lambda^*$ in terms of the model parameters. Formulate the linear-quadratic approximation for the decision model. Derive analytical expressions for the approximate shadow price and optimal policy function coefficients in terms of the model parameters.

(c) Develop a computer program that will compute an approximate optimal policy over the interval $S = [4, 8]$ using Chebychev polynomial projection. The degree of the interpolating polynomial $n$ should be treated as parameters by the program. Solve the model using $n = 2$, $n = 10$, and $n = 50$.

(d) Using the graphics package of your choice, graph the optimal policies derived in (a), (b), and (c) together. To draw your graph, evaluate the policy functions at 101 equally points over the interval $[4, ]$.

(e) Repeat (a)-(d), except now assume that the resource is owned by a profit maximizing monopolist, rather than a benevolent social planner.

9.5. Consider the commodity storage model of section 5.4, except now assume that harvest $h_{t+1}$ at the beginning of year $t + 1$ is the product of the acreage $a_t$ planted in year $t$ and a random yield $y_{t+1}$ realized at the beginning of year $t+1$:

$$h_{t+1} = a_t \cdot y_{t+1}.$$

Further assume that acreage planted is a function

$$a_t = (E_t p_{t+1})^{0.8}$$

of the price expected to prevail at harvest time conditional on the information known at planting time and that the $\log y_t$ are serially independent and normally distributed with mean 0 and standard deviation 0.2.

(a) Write the conditions that characterize the rational expectations equilibrium for this market in terms of the $\lambda$ solution function to be computed.

(b) Develop a computer program to solve for the rational expectations equilibrium using Chebychev polynomial projection methods.

(c) Graph acreage planted in terms of the supply available at the beginning of the period.

(d) Estimate the steady-state mean and variance of acreage planted using Monte-Carlo simulation. A five thousand year simulation will be adequate.

Use an appropriate 5 point discretization for the random yield and set the minimum and maximum supply levels to 0.6 and 2.0.

9.6. 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 treating the state space as continuous. Approximate the value function using a linear spline basis with nodes spaced one unit apart on the interval $[0, 100]$

(a) Graph the shadow price as a function of the stock level.

(b) Graph the optimal harvest as a function of the stock level.

(c) Plot optimal stock levels through year 20, beginning with a stock of 100.

(d) Plot optimal harvests through year 20, beginning with a stock of 100 at time 0.

9.7. A social planner wishes to maximize the present value of current and future net social welfare derived from industrial production over an infinite time horizon. Net social welfare in period $t$ is

$$\alpha_0 + \alpha_1 q_t - 0.5 q_t^2 - c s_t$$

where $q_t$ is industrial production in period $t$ and $s_t$ is the pollution level at the beginning of period $t$. The pollutant stock is related to industrial production as follows:

$$s_{t+1} = s_t^{\gamma} + q_t.$$

Assume $\alpha_0 = 2.0$, $\alpha_1 = 3.0$, $\gamma = 0.6$, $c = 1.0$, and $\delta = 0.9$.

Solve this problem using linear quadratic approximation and collocation, treating the state space as continuous, over the interval $[2, 7]$. Use a 10 degree Chebychev polynomial basis for the collocation scheme:

(a) Compute and plot, on one figure, the optimal shadow price function obtained by L-Q approximation and by collocation.

(b) Compute and plot, on one figure, the optimal production policy function obtained by L-Q approximation and by collocation.

(c) Plot pollution level through year 20, beginning with a pollution level of 7 at time 0.

(d) Plot production through year 20, beginning with a pollution level of 7 at time 0.

9.8. In a widely cited article, Deaton and Larocque pose a time-stationary, discrete-time dynamic model of a market for a storable primary commodity in which:

- consumption $c$ in any period is a deterministic function $c = D(p)$ of price $p$;
- storage $x$ is costless and undertaken by risk neutral, expected profit maximizers who discount the future at a constant per-period rate $r$
- new production $\tilde{h}$ at the beginning of each period is exogenous and random.

Please answer the following questions:

(a) Formulate and interpret the intertemporal arbitrage condition that must be satisfied by prices in this commodity market.

(b) How would you solve and simulate this model in order to gain an understanding of price dynamics in this commodity market.

9.9. In a well-known article on primary commodity price dynamics, Deaton and Larocque pose the functional equation

$$f(x) = \delta E_h \max\{\ q^{-1}(x)\ ,\ f(x - q(f(x)) + h)\ \} \qquad x \in X$$

where $\delta$ is a known discount factor, $q(\cdot)$ is a known demand function, $h$ is a random variable with known continuous distribution, and $f$ is an unknown function that gives the equilibrium market price in terms of the available supply $x$. Describe the steps that you would take to solve this functional equation numerically for $f$. Also discuss how you would test the validity of your approximation.

9.10. The Bellman equation of a deterministic autonomous continuous-time dynamic optimization model takes the form

$$-rV(s) = \max_x \{f(s,x) + V'(s)g(s,x)\} \qquad s \in S$$

where $f$ and $g$ are known functions, $r$ is the interest rate, and $S \subset \Re$ is a compact interval. Describe the steps that you would take to solve this functional equation numerically for the unknown value function $V$. Also discuss how you would test the validity of your approximation.

9.11. Consider a infinitely-lived worker searching for a job. At each point in time $t$, the worker observes a per-period wage offer $w_t$. The worker may accept the offer, committing him to receive that per-period wage thereafter in perpetuity. Alternatively, he may reject the offer, earning nothing in the current period, and wait for a hopefully better wage offer the following period. The wage offers follow an autoregressive process

$$\log(w_{t+1}) = 0.5 \log(w_t) + \epsilon_t$$

where the $\epsilon_t$ are i.i.d. normal with mean zero and standard deviation 0.2. The worker's objective is to maximize the present value of current and expected future wages using a discount factor of 0.9 per period.

(a) Formulate Bellman's equation for the worker's optimization problem.

(b) Solve Bellman's equation for the worker's value function using a continuous-state numerical collocation scheme of your choosing.

(c) Plot the value function.

(d) Plot the residual function.

(e) Estimate the worker's reservation wage, that is, the minimum wage he would accept.

9.12. A farmer wishes to maximize the present value of current and future profits over an infinite horizon assuming a stable corn price of \$2.00 per bushel, a stable commercial fertilizer cost of \$0.25 per pound, and an annual discount factor of 0.9. The farmer's corn yield in year $t$, in bushels per acre, is given by the Mitscherlick-Baule production function

$$y_t = 140[1 - 0.3\exp(-0.1s_t)][1 - 0.1\exp(-1.3x_t)]$$

where $s_t$ is soil fertilizer carryin and $x_t$ is fresh fertilizer applied topically at planting time, both measured in pounds per acre. The fertilizer carryover dynamic is

$$s_{t+1} = 9.0 + 0.7s_t + 0.1x_t$$

Solve Bellman's equation for the value function over the interval $[15, 60]$ using both linear-quadratic approximation and a 10 degree Chebychev collocation scheme.

(a) Plot of the shadow price function produced by both the L-Q and Chebychev approximations.

(b) Plot the optimal fertilizer application policy produced by both the L-Q and Chebychev approximations.

(c) Plot the residual function for the Chebychev approximation.

(d) Plot carryin through year 20, beginning with a carryin of 15 at time 0. (Use Chebychev approximant.)

(e) Plot fertilizer applications through year 20, beginning with a carryin of 15 at time 0. (Use the Chebychev approximant.)

# Chapter 10

# Continuous Time Models: Theory and Examples

In this chapter we discuss models that treat time as a continuum. Such models are typically expressed in terms of differential equations, either ordinary or partial. Our discussion proceeds in three sections. First, we discuss models of asset prices that are based on arbitrage considerations alone and that do not depend on solving a decision problem. Many financial asset, including bonds, futures and some options are in this class. We then take up the topic of stochastic control, i.e., of optimal decision making applied to processes that evolve continuously in time. Such problems will be illustrated with examples of growth models, portfolio choice and resource management. Next, we will turn to problems involving free boundaries, which arise when a discrete choice is made. Examples of such problems include entry/exit decisions, option exercise and asset replacement.

Continuous time models make extensive use of Ito processes, which are continuous time Markov processes. Because Ito processes do not possess time derivatives, it is necessary to make use of stochastic calculus. Especially useful is the extension of the chain rule known as Ito's Lemma and the relationships between expectations and differential equations embodied in so-called forward and backward equations and the Feynman-Kac Equation. A review of these topics is provided in the Mathematical Appendix; more details can be found in the references discussed at the end of this chapter.

## 10.1 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 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.

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[1]

$$dS = \mu_S dt + \sigma_S dz.$$

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 $\delta_V$ and $\delta_W$.

One can create a portfolio consisting of one unit of $V$ and $h$ units of $W$, the value of which is described by

$$dV + hdW = [\mu_V + h\mu_W]dt + [\sigma_V + h\sigma_W]dz.$$

This portfolio can be made risk free by the appropriate choice of $h$, specifically by setting the $dz$ term to 0:

$$h = -\sigma_V/\sigma_W.$$

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

$$\frac{\mu_V + \delta_V - rV}{\sigma_V} = \frac{\mu_W + \delta_W - rW}{\sigma_W}.$$

This expression must hold or any assets that depend on $S$ and therefore both sides must equal a function $\theta(S,t)$, that does not depend on the specific features of the

[1]The following notational conventions are used. $\mu$, $\sigma$ and $\delta$ 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.

particular derivative asset. In other words, the function $\theta$ is common to all assets whose values depend on $S$. $\theta$ 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 therefore 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 the risk free return plus a risk adjustment, $rV + \theta\sigma_V$.

Ito's Lemma provides a way to evaluate the $\mu_V$ and $\sigma_V$ 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

$$rV = \delta_V + V_t + (\mu_S - \theta\sigma_S)V_S + \tfrac{1}{2}\sigma_S^2 V_{SS}. \tag{10.1}$$

This is the fundamental differential equation that any asset derived from $S$ must satisfy, in the sense that it must be satisfied by any frictionless economy in equilibrium. This is a remarkable result. Its says that all assets that depend on $S$ satisfy a linear PDE that is identical in its homogeneous part. Assets are differentiated only by the forcing term $\delta_V$ and by boundary conditions.

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

$$\mu_S - \theta\sigma_S = rS - \delta_S.$$

The value of any asset, $V$, which is derived from $S$, therefore 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, $\theta$, can be inferred from the behavior of $W$:

$$\theta(S,t) = \frac{\mu_W + \delta_W - rW}{\sigma_W},$$

where $\delta_W$ is the dividend flow acquired by holding $W$.

The no-arbitrage condition provides a very convenient framework for pricing a wide variety of financial assets. One must specify the nature of the state process ($\mu$ and $\sigma$) and how the asset and interest rate depend on the state. Any asset that pays a state-dependent return at a fixed terminal date $T$ can then (in principle) be valued. With a little more work we will also be able to value assets that may be terminated early, such as American style option and callable bonds. Such assets, however, entail an optimal termination choice; we will return to them in Section 10.3.3 after we discuss control theory in continuous time.

**Example: Bond Pricing**
Suppose that the instantaneous rate of interest is described by the (risk-neutral) process

$$dr = \mu(r)dt + \sigma(r)dz.$$

A bond paying 1 unit of account at time $T$ has a current price, $B(r,t;T)$, that satisfies the arbitrage condition

$$rB = B_t + \mu(r)B_r + \tfrac{1}{2}\sigma^2(r)B_{rr},$$

subject to the boundary condition at time $T$ that $B(r,T;T) = 1$. Specific examples are left as exercises.

**Example: Black-Scholes Formula**
Consider a non-dividend paying (or payout protected) stock ($\delta_S = 0$), the price of which follows

$$dS = \mu S dt + \sigma S dz,$$

where $\mu$ and $\sigma$ 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 $\sigma^2\Delta t$ (see Appendix A, Section ??).

The stock as itself an asset with no flow of payments and hence must satisfy the arbitrage condition that

$$\mu(S) - \theta\sigma(S) = rS.$$

A derivative asset that depends on $S$ and that generates a one-time return at time $T$ therefore has value, $V(S,t)$, that satisfies the arbitrage condition

$$rV = V_t + rSV_S + \tfrac{1}{2}\sigma^2 S^2 V_{SS}.$$

A European call option on $S$ with a strike price of $K$ has a payout at time $T$ of $S-K$ if $S > K$ and 0 otherwise. The boundary condition for the PDE is, therefore, that $V(S,T) = \max(0, S-K)$.

The value of such an option is

$$V(S,t) = S\Phi(d) - e^{-r\tau}K\Phi\left(d - \sigma\sqrt{\tau}\right)$$

where $\tau = T - t$,

$$d = \frac{\ln(S/K) + r\tau}{\sigma\sqrt{\tau}} + \tfrac{1}{2}\sigma\sqrt{\tau},$$

and $\Phi$ is the standard normal CDF:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-\frac{1}{2}z^2}dz.$$

Some tedious manipulations will demonstrate that

$$V_S = \Phi(d),$$

$$V_{SS} = \frac{\phi(d)}{\sigma S\sqrt{\tau}}$$

and

$$V_t = -\frac{\sigma S\phi(d)}{2\sqrt{\tau}} - re^{-r\tau}K\phi\left(d - \frac{\sigma\sqrt{\tau}}{2}\right),$$

where

$$\phi(x) = \Phi'(x) = \frac{e^{-\frac{1}{2}x^2}}{\sqrt{2\pi}}$$

(the partial derivatives are known as the delta, gamma and theta of the call option and are used in hedging). Using these expressions it is straightforward to verify that the partial differential equation above, including the boundary condition, is satisfied.

**Example: Exotic Options**

The basic no-arbitrage approach can be applied to more complicated derivative assets. We illustrate with several types of so-called exotic options, Asian, lookback and barrier options.

An Asian option is one for which the payout depends on the average price of the underlying asset over some pre-specified period. There are two basic types: the first has a strike price equal to the average and the second pays the positive difference between the average price and a fixed strike price.

Defining $S$ to be the underlying price and letting the Asian option depend on the average price over the period 0 to $T$ its expiration date, the relevant average is

$$A = \frac{1}{T}\int_0^T S_t dt.$$

The average strike Asian call option pays, at time $T$, $\max(S - A, 0)$ and the fixed strike Asian call option, with strike price $K$, pays $\max(A - K, 0)$.

Suppose the dynamics of $S$ are given by

$$dS = \mu(S)dt + \sigma(S)dW.$$

It is not enough, however, to know current $S$ because the average depends on the path $S$ takes; in other words the option is not Markov in $S$. We can, however, expand the state space by defining

$$C_t = \int_0^t S_\tau d\tau.$$

The option's value will depend on $S$, $C$ and $t$. Noting that $dC = Sdt$, the option satisfies the usual no-arbitrage condition

$$rV = V_t + \mu(S)V_S + SV_C + \tfrac{1}{2}\sigma^2(S)V_{SS}$$

with the terminal value equal to

$$V(S, C, T) = \max(S - C/T, 0)$$

or

$$V(S, C, T) = \max(C/T - K, 0)$$

for average and fixed strike Asian, respectively.

In the special case that $S$ evolves according to a geometric Brownian motion (the assumption made to derive the Black-Scholes formula) this two dimensional PDF can be simplified. We use a guess and verify strategy by defining the transformation $y = S/C$ and the guess that $V(S, C, \tau)$ has the form $Cv(S, \tau)$. This will work for the average strike option, with the terminal condition $v(y, T) = \max(y - 1/T, 0)$.[2] The partial derivatives are $V_\tau = Cv_\tau$, $V_S = v_y$, $V_{SS} = v_{yy}/C$ and $V_C = v - Cv_y(S/C^2) =$

[2]The terminal condition for the fixed strike Asian cannot be put in this form so it is clear that its value is not proportional to $C$. A closed form expression exists for a fixed strike Asian when the average is defined as $\exp\left(\int_0^T Sdt/T\right)$ (geometric average), however; we leave this as an exercise.

$v - yv_y$. These can be substituted into the no-arbitrage condition to derive the expression

$$rCv = Cv_t + ryCv_y + yC(v - yv_y) + \tfrac{1}{2}\sigma^2 y^2 Cv_{yy}.$$

Notice that $C$ is a common term and can be divided out, leaving

$$(r - y)v = v_t + (r - y)yv_y + \tfrac{1}{2}\sigma^2 y^2 v_{yy}.$$

Instead of a two dimensional PDE, we only have to solve a one-dimensional one; this is far easier to accomplish numerically.

A lookback option is one written on the maximum price over the option's life and can be either a lookback strike or a fixed strike lookback. Like Asian options, one must define an additional state variable to keep track of the maximum price. Let

$$M_t = \max_{\tau \in [0,t]} S_\tau.$$

The terminal conditions are $V(S, M, T) = \max(M - K, 0)$ for the fixed strike lookback call and $V(S, M, 0) = \max(M - S, 0)$ for the lookback strike put.

Notice that $dM = 0$ for $S < M$. Hence the no-arbitrage condition for $S < M$ does not involve $M$:

$$rV = V_t + \mu(S)dt + \tfrac{1}{2}\sigma^2(S)dW.$$

This does not mean, however, that $V$ doesn't depend on $M$. At the point $S = M$ the option value must satisfy $V_M(M, M, t) = 0$.

Finally, we consider one of the many types of barrier options, the so-called down-and-out option. This option behaves like a normal put option so long as the underlying price stays above some prespecified barrier $B$. However, if the price hits $B$ anytime during the life of the option, the option is immediately terminated with some rebate $R$ paid to its holder.

Down-and-out options satisfies the usual no-arbitrage condition for $S \in [B, \infty)$. In addition to the usual terminal boundary condition, $V(S, T) = \max(K - S, 0)$, this additional boundary condition $S(B, t) = R$ must be imposed.

A number of other exotic options exist that, like American style options, have a value that depends on the holder optimally making some decision during the life of the option. For example, a shout option allows the holder to lock in a minimal return at some point during the option's life so the return (at time $T$) is determined by solving

$$\max_{\tau \in [0,T]} max(S_\tau - K, S_t - K).$$

Many options and other assets also have compound features; their return is based on the value of another option or other derivative asset. For example, bond options and futures options depend on the value of a specified bond or on a specified futures price at the expiration of the option. To price these, one must first determine the value of the underlying and use this value as a terminal condition for the option.

Consider, for example, an option written on a 3-month Treasury bond. The value of the Treasury bond satisfies the no-arbitrage condition with $B(S, T + 0.25) = 1$ as a terminal condition. The terminal condition for a call option with a strike price of $K$ is $V(S, T) = B(S, T)$. Using this approach, compound options of considerable complexity can be valued. In general there will not be closed form solutions for such assets, but it is relatively easy to price them numerically, as we will see in the next chapter.

**Example: Multivariate Affine Asset Pricing Model**

As the dimension of the state process increases, the use of the no-arbitrage PDE becomes increasingly difficult to apply, as we shall see in the next chapter. There are some cases, however, for which this so-called curse of dimensionality can be avoided. The most important case is the affine asset pricing model, which has been widely applied, especially in modeling interest rate and futures price term structure.

Suppose that the risk-neutral state price process can be described by an affine diffusion, which takes the form

$$dS = (a + AS)dt + C\text{diag}\left(\sqrt{b + BS}\right) dW,$$

where $a$ and $b$ are $n \times 1$ and $A$, $B$ and $C$ are $n \times n$ (the $\sqrt{\ }$ operator is applied element by element). Furthermore, the risk free interest rate is an affine function of the state, $r_0 + rS$ ($r_0$ is a scalar and $r$ is $1 \times n$) and the log of the terminal value of the asset is an affine function of the state, $\ln(V(S, 0)) = h_0 + hS$.

Given these assumptions, it is straightforward to show that the log of the asset value is affine in the state, with the coefficients depending on the time-to-maturity ($\tau = T - t$):

$$V(S, \tau) = \exp(\beta_0(\tau) + \beta(\tau)S).$$

It is clear this satisfies the terminal condition when $\beta_0(0) = h_0$ and $\beta(0) = h$. Substituting the proposed value of $V$ into the no-arbitrage condition yields

$$\begin{aligned}(r_0 + rS)V &= -(\beta_0'(\tau) + \beta'(\tau)S)V + \beta(\tau)(a + AS)V \\ &\quad + \tfrac{1}{2}\text{trace}\left(C\text{diag}(b + BS)C^\top\beta(\tau)^\top\beta(\tau)\right)V.\end{aligned}$$

$V$ is a common term that can be divided out and the remaining expression is affine in $S$. The expression is therefore satisfied when[3]

$$\beta_0'(\tau) = \beta(\tau)a + \tfrac{1}{2}\beta(\tau)C\text{diag}(C\beta(\tau))b - r_0$$

and

$$\beta'(\tau) = \beta(\tau)A + \tfrac{1}{2}\beta(\tau)C\text{diag}(C\beta(\tau))B - r.$$

The $n+1$ coefficient functions $\beta_0(\tau)$ and $\beta(\tau)$ are thus solutions to a system of ordinary differential equations, which are easily solved, even when $n$ is quite large.

## 10.2 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[4]

$$\max_{x(S,t)} \int_0^T e^{-\rho t} f(S,x)dt + e^{-\rho T} R(S(T)), \text{ s.t. } dS = g(S,x)dt,$$

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 10.3 on free boundaries).

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 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

[3] We use the facts that $\text{trace}(xyz) = \text{trace}(zxy)$ and, when $x$ and $y$ are vectors, $\text{diag}(x)y = \text{diag}(y)x$.

[4] We cover here the more common discounted time autonomous problem. The more general case is developed as an exercise.

define the state variable, $S$, to be a controllable Ito process, meaning that the control variable, $x$, influences the value of the state:[5]

$$dS = g(S,x)dt + \sigma(S)dz.$$

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:

$$\rho V(S,t) = \max_x \left( f(S,x,t) + \frac{E_t dV(S,t)}{dt} \right). \tag{10.2}$$

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 $V$ ($\rho V$) 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.[6]

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:[7]

$$\rho V = \max_x f(S,x) + V_t + g(S,x)V_S + \tfrac{1}{2}\sigma^2(S)V_{SS}. \tag{10.3}$$

The maximization problem is solved in the usual way by setting the first derivative equal to zero:

$$f_x(S,x) + g_x(S,x)V_S = 0. \tag{10.4}$$

[5] 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.

[6] It is important to note that the arbitrage interpretation requires that the discount rate, $\rho$, be appropriately chosen (see Section 10.1 for further discussion).

[7] Also known as the Hamilton-Jacobi-Bellman equation.

leading (in principle) to a solution of the form

$$x = x(S, V_S)$$

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 management exercise on page 386). The optimal control can be combined with (10.3) to form the *concentrated Bellman equation*:

$$\rho V = f(S, x(S, V_S)) + V_t + g(S, x(S, V_S))V_S + \tfrac{1}{2}\sigma^2(S)V_{SS}, \tag{10.5}$$

which must be solved for $V(S)$.

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 dynamic problem into a deterministic one.

In finite time horizon problems, the value function is a function of time and the time derivative $V_t$ appears in the Bellman's equation. 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

$$\rho V = \max_x f(S, x) + g(S, x)V_S + \tfrac{1}{2}\sigma^2(S)V_{SS}.$$

### 10.2.1 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 $P = 0$, meaning if the process hits the barrier it stays there forever (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: $V(I, P)$. 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 345.

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 $V_{SS}(\underline{S},R) = V_{SS}(\overline{S},R) = 0$.

Like all rules of thumb, however, there are exceptions. The exceptions tend to arise in problems involving singular processes for which 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 boundary for this process, meaning that $S(t) \geq 0$ with probability 1. 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. Several examples we will discuss have singular boundary conditions.

### 10.2.2 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 section 10.1 can be applied fruitfully 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 that should be used to value an investment project. 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, (10.1) must hold. Together these relationships imply that

$$\rho = r + \theta\sigma_S V_S/V = r + \theta\sigma_V/V \tag{10.6}$$

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, if $S$ is the value of a traded asset,

$$\hat{\mu}_S = rS - \delta_S.$$

Which form is more useful depends on whether it is easier to obtain estimates of the market price of risk for $S$, $\theta_S$, or income stream generated by $S$, $\delta_S$.

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, $\rho$, therefore implicitly determines the market price of risk (using 10.6).

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 $\theta$ or $\rho$ must be determined by other means.

When $S$ is influenced by the control $x$, 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:

$$\text{A)} \quad rV = \max_x f(S,x,t) + V_t + (rS - \delta_S)\, V_S + \tfrac{1}{2}\sigma^2(S,t) V_{SS}$$

$$\text{B)} \quad rV = \max_x f(S,x,t) + V_t + (g(S,x,t) - \theta\sigma(S,t))\, V_S + \tfrac{1}{2}\sigma^2(S,t) V_{SS}$$

$$\text{C)} \quad \rho V = \max_x f(S,x,t) + V_t + g(S,x,t) V_S + \tfrac{1}{2}\sigma^2(S,t) V_{SS}$$

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. 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 $\theta$ or for $\rho$.

### 10.2.3 Euler Equation Methods

As in the discrete time case, it may be possible to eliminate the value function (or costate variable) and express the optimality conditions in terms of the state and control alone (see discussion for discrete time in Section ??). Such an expression is known as an Euler equation. In the discrete time case, this was most useful in problems for which $g_S(x, S, \epsilon) = 0$. In the continuous time case, however, Euler equation methods are most useful in deterministic problems. As before, we discuss the infinite horizon case and leave the reader to work out the details for the finite horizon case.

Suppose

$$dS = g(x, S)dt.$$

The Bellman Equation is

$$\rho V(S) = \max_x f(x, S) + g(x, S)V'(S),$$

with FOC

$$f_x(x, S) + g_x(x, S)V'(S) = 0.$$

Let $h(x, S) = -f_x(x, S)/g_x(x, S)$, so the FOC can be written

$$V'(S) = h(x, S). \tag{10.7}$$

Using the Envelope Theorem applied to the Bellman Equation,

$$(\rho - g_S(x, S))V'(S) - f_S(x, S) = g(x, S)V''(S).$$

Using $h$ and its total derivative with respect to $S$:

$$V''(S) = h_S(x, S) + h_x(x, S)\frac{dx}{dS},$$

the terms involving $V$ can be eliminated:

$$(\rho - g_S(x, S))h(x, S) - f_S(x, S) = g(x, S)\left[h_S(x, S) + h_x(x, S)\frac{dx}{dS}\right]. \tag{10.8}$$

This is a first-order differential equation that can be solved for the optimal feedback rule, $x(S)$. The "boundary" condition is that the solution pass through the steady state at which $dS/dt = 0$ and $dx/dt = 0$. The first of these conditions is that $g(x, S) = 0$, which in turn implies that the left hand side of (10.8) equals 0:

$$(\rho - g_S(x, S))h(x, S) - f_S(x, S) = 0.$$

These two equations are solved simultaneously (either explicitly or numerically) to yield boundary conditions for the Euler Equation. This approach is applied in the example beginning on page 353.

### 10.2.4 Examples

**Example: Optimal Renewable Resource Extraction**
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. The marginal cost of harvesting the resource 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

$$f(S,q) = \int_0^q D^{-1}(z)dz - c(S)q,$$

where $D$ is the demand function for the resource.

With a discount rate of $\rho$, 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 the 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 the problem operational, it must be parameterized. Specifically the biological growth function is

$$B(S) = \frac{\alpha}{\beta} S(1 - (S/K)^\beta),$$

the demand function is

$$D(p) = bp^{-\eta}$$

and the marginal cost function is

$$c(S) = cS^{-\gamma}.$$

In general this model must be solved numerically, but special cases do exist which admit an explicit solution. Specifically, if $\gamma = 1 + \beta$ and $\eta = 1/(1 + \beta)$ the value function has the form

$$V(S) = -\phi\left(\frac{1}{S^\beta} + \frac{\alpha}{\rho K^\beta}\right)$$

where $\phi$ 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 $\phi$ using a standard root finding solver (see Chapter 3) and for some values of $\beta$ a complete solution is possible. Table 10.1 provides three special cases discussed by Pindyck that have closed form solutions, including the limiting case as $\beta \to 0$.

**Example: Stochastic Growth**
An economy is characterized by a function describing the productivity of capital, $K$, that depends, both in mean and variance, on an exogenous technology shock, denoted $Y$. $Y$ is governed by

$$dY = \kappa(1 - Y)dt + \sigma\sqrt{Y}dz.$$

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 $\rho$.

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

$$dEY = (aEY - b)dt.$$

**Table 10.1: Known Solutions to the Optimal Harvesting Problem**

| $\beta$ | $\eta$ | $\gamma$ | $B(S)$ | $V(S)$ | $V_S(S)$ | $q^*(S)$ |
|---|---|---|---|---|---|---|
| 1 | $\frac{1}{2}$ | 2 | $\alpha S(1-S/K)$ | $-\phi_1\left(\frac{1}{S}+\frac{\alpha}{\rho K}\right)$ | $\frac{\phi_1}{S^2}$ | $\frac{b}{\sqrt{c+\phi_1}}S$ |
| 0 | 1 | 1 | $\alpha S\ln(K/S)$ | $\frac{b}{\alpha+\rho}\ln(S)+\phi_2$ | $\frac{b}{(\alpha+\rho)S}$ | $b\theta S$ |
| $-\frac{1}{2}$ | 2 | $\frac{1}{2}$ | $2\alpha S\left(\sqrt{\frac{K}{S}}-1\right)$ | $-\phi_3\left(\sqrt{S}+\frac{\alpha\sqrt{K}}{\rho}\right)$ | $-\frac{\phi_3}{2\sqrt{S}}$ | $\frac{b}{(c-\phi_3/2)^2}S$ |

where

$$\phi_1 = 2\left(\frac{b}{\alpha+\rho-\sigma^2}\right)^2\left(1+\sqrt{1+c\left(\frac{\alpha+\rho-\sigma^2}{b}\right)^2}\right)$$

$$\phi_2 = \frac{b}{\rho}\left(\ln(b\theta)-c\theta+\frac{1}{\alpha+\rho}\left(\alpha\ln(K)-b\theta-\tfrac{1}{2}\sigma^2\right)\right),\ \theta=\frac{\alpha+\rho}{b+c(\alpha+\rho)}$$

$$\phi_3 = c-\sqrt{c^2+\frac{2b}{\alpha+\rho+\sigma^2/8}}$$

This differential equation can be solved for the expected value of $Y$:

$$E_t Y_T = (Y_t - b/a)e^{a(T-t)} + b/a.$$

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 Bellman Equation for this problem is

$$\begin{aligned}\rho V = \max_c \ln(c) &+ V_K(\beta KY - c) + V_Y(aY - b) \\ &+ \tfrac{1}{2}V_{KK}\epsilon^2 K^2 Y + \tfrac{1}{2}V_{YY}\sigma^2 Y + V_{KY}\sigma\epsilon KY\end{aligned}$$

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 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:

$$\begin{aligned}\rho\left(\tfrac{\ln(K)}{\alpha} + f(Y)\right) = \\ \ln(\alpha K) + \left(\tfrac{\beta}{\alpha}Y - 1\right) + f'(Y)(aY - b) - \tfrac{\epsilon^2 Y}{2\alpha} + \tfrac{1}{2}f''(Y)\sigma^2 Y.\end{aligned}$$

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

$$V(K,Y) = E\left[\int_0^\infty e^{-\rho t}\ln(\rho K)dt\right] = \frac{\ln(\rho)}{\rho} + \int_0^\infty e^{-\rho t}E\left[\ln(K)\right]dt \tag{10.9}$$

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 (10.9)[8]

$$\begin{aligned} V(K,Y) &= \int_0^\infty e^{-\rho t} E\left[\ln(K)\right] dt + \frac{\ln(\rho)}{\rho} \\ &= \int_0^\infty \left(\left(\ln(K_0) - c_0\right) e^{-\rho t} + c_0 e^{(a-\rho)t} + c_1 t e^{-\rho t}\right) dt + \frac{\ln(\rho)}{\rho} \\ &= \frac{\ln(K_0) - c_0}{\rho} + \frac{c_0}{\rho - a} + \frac{c_1}{\rho^2} + \frac{\ln(\rho)}{\rho}. \end{aligned}$$

Substituting in the values of $c_0$ and $c_1$ and rearranging we obtain an expression for the value function

$$V(K,Y) = \frac{\ln(K)}{\rho} + \frac{\beta - \frac{1}{2}\epsilon^2}{\rho(\rho - a)} Y + \frac{1}{\rho}\left(\ln(\rho) - \frac{b(\beta - \frac{1}{2}\epsilon^2)}{\rho(\rho - a)} - 1\right).$$

[8] If the third line is problematic for you it might help to note that

$$\int t e^{-\rho t} dt = -\frac{e^{-\rho t}}{\rho}\left(t + \frac{1}{\rho}\right).$$

(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 transforms a problem that is essentially impossible to solve in general into one that is relatively straightforward to solve. 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

$$dP = \mu(P)dt + \sigma(P)dz,$$

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.[9] A portfolio can be defined by the 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

$$dW = \sum_{i=1}^{n} N_i dP_i$$

[9] The case in which a riskless asset is available is treated in an exercise.

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 $\mu(P) = P\mu$ and $\sigma(P) = P\sigma$ (where $\mu$ and $\sigma$ are constants), implying that:

$$dW/W = w^\top \mu dt + w^\top \sigma dz,$$

i.e., $W$ 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

$$dW = \left[Ww^\top \mu - C\right] dt + Ww^\top \sigma dz.$$

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 $\rho$. The Bellman's Equation for this problem is[10]

$$\rho V = \max_{C,w} U(C) + \left[Ww^\top \mu - C\right] V_W + \tfrac{1}{2} W^2 w^\top \Sigma w V_{WW},$$

s.t. $\sum_i w_i = 1$.

The FOC associated with this maximization problem are

$$U'(C) = V_W, \tag{10a}$$

$$WV_W \mu + W^2 V_{WW} \Sigma w - \lambda \underline{1} = 0, \tag{10b}$$

and

$$\sum_i w_i = 1, \tag{10c}$$

where $\lambda$ is a Lagrange multiplier introduced to handle the adding-up constraint on the $w_i$. A bit of linear algebra applied to (10b) and (10c) will demonstrate that the

[10] If prices were not geometric Brownian motion the coefficients $\mu$ and $\sigma$ 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.

optimal portfolio weight vector, $w$, can be written as a linear combination of vectors, $\nu$ and $\theta$, that are independent of the investor's preferences:

$$w = \nu + \alpha(W)\theta, \tag{11}$$

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

$$\alpha(W) = -\frac{V_W}{WV_{WW}},$$

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 aversion rises (as $\alpha(W)$ decreases). This is understandable when it is noticed that the first portfolio is the minimum risk portfolio, i.e., $\nu$ 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

$$\alpha(W) = -\frac{V_W(W)}{WV_{WW}(W)}.$$

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.

**Example: Neoclassical Growth Model**

Ramsey introduced what has become a standard starting place for studying optimal economic growth. The basic model has been refined and extended in numerous ways. We present here simple version.[11] A single (aggregate) good economy is governed by a production technology, $f$. The net output from the production process depends on the level of the capital stock $K$. That output can either be consumed at rate $C$ or invested, thereby increasing the capital stock. A social utility function depends on the rate of consumption, $U(C)$; a social planner attempts to maximize the discounted stream of social utility over an infinite time horizon, using a constant discount factor $\rho$.

The optimization problem can be expressed in terms of $K$, the state variable, and $C$, the control variable, as follows

$$\max_{C(t)} \int_0^\infty e^{-\rho t} U(C) dt,$$

subject to the state transition function $K' = q(K) - C$.

This is a deterministic infinite time problem so $V_{KK}$ and $V_t$ do not enter the Bellman's equation. The Bellman's equation for this problem is

$$\rho V(K) = \max_C U(C) + V'(K)\left(q(K) - C\right)$$

The maximization problem requires that

$$U'(C) = V'(K). \tag{12}$$

We can derive an Euler equation form for this problem by eliminating the value function, thereby obtaining a differential equation for consumption in terms of current capital stock. Applying the Envelope Theorem,

$$\rho V'(K) = q'(K)V'(K) + [q(K) - C]V''(K). \tag{13}$$

Combining (12), (13) and the fact that $V''(K) = U^{''}(C)C'(K)$ yields

$$-\frac{U'(C)}{U''(C)}(q'(K) - \rho) = (q(K) - C)C'(K).$$

Thus the optimal decision rule $C(K)$ solves a first order differential equation.

[11] We alter Ramsey's original formulation by including a discount factor in the optimization problem, as is standard in modern treatments.

The boundary condition for this differential equation is that the solution passes through the point $(K^*, C^*)$ which simultaneously solves $dK/dt = 0$ and the Euler condition:

$$q'(K^*) = \rho$$
$$C^* = q(K^*).$$

**Example: Non-Renewable Resource Management**

A firm that manages a non-renewable resource obtains a net return flow of $Ax^{1-\alpha}$ per unit of time, where $x$ is the rate at which the resource is extracted. The stock of the resource is governed by

$$dS = -xdt.$$

The extraction rate is bounded below by 0 ($x \geq 0$) and constrained to equal 0 if the stock is 0 ($S = 0 \Rightarrow x = 0$). The manager seeks to solve

$$V(S) = \max_x E\left[\int_0^\infty e^{-\rho\tau} A x_\tau^{1-\alpha} d\tau\right].$$

The Bellman's equation for the problem is

$$\rho V(S) = \max_x Ax^{1-\alpha} - xV_s.$$

The boundary condition is that $V(0) = 0$. The first order optimality condition is

$$V_S(S) = (1-\alpha)Ax^{-\alpha}.$$

Using an Euler equation approach, apply the envelope theorem to obtain

$$\rho V_S = -xV_{SS}.$$

We have an expression for $V_S$ from the optimality condition and this can be differentiated with respect to $S$ to obtain

$$\rho(1-\alpha)Ax^{-\alpha} = \alpha(1-\alpha)Ax^{-\alpha}\frac{dx}{dS}.$$

Simplifying, this yields

$$\frac{dx}{dS} = \frac{\rho}{\alpha},$$

which, with the boundary condition that $x = 0$ when $S = 0$, produces the optimal control

$$x(S) = \frac{\rho}{\alpha}S$$

and the value function

$$V(S) = \left(\frac{\alpha}{\rho}\right)^\alpha AS^{1-\alpha}.$$

## 10.3 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 which must be determined as part of the solution. Such problems are known as free boundary problems. The boundary marks the location where some discrete action is taken, generally taking the form of effecting an instantaneous change in the value of a continuous or a discrete state.[12]

Table 10.2 contains a classification of different free boundary problems that have appeared in the economics literature. An 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 state variable 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 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 move it to some point below the boundary (impulse control).

**Table 10.2: Types of Free Boundary Problems**

| BARRIERS: | |
|---|---|
| Problem | Action at Boundary |
| Impulse control | Jump from trigger to target |
| Barrier control | Move along boundary |
| | |
| TRANSITIONAL BOUNDARIES: | |
| Problem | Action at Boundary |
| Discrete states | Change state |
| Bang-bang | Switch between control extrema |

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

[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.

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.

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 evaluated using an arbitrary boundary and the value function using 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 Appendix A, Section A.5.3). The optimal choice of the boundary must then add additional restrictions that ensure its optimality. We therefore distinguish in Table 10.2 between a point, $S^a$, 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 then checking whether the optimality condition holds.

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 arbitrary controls.

For optimality we will require an additional boundary condition for each possible binary choice. The additional constraints can be derived formally by maximizing the value 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.

### 10.3.1 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 $\Delta S$.

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

[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

cannot change quickly; essentially the size of the change in the state must be small if the time interval over which the change is measured is 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 non-negative. 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 processes, 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 points in time. An impulse control strategy would be optimal when there are positive 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. 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 and must also decide 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 instantaneously so there is no reason not to let inventory fall to zero). A third possibility 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.

**Example: Asset Replacement**
Consider the problem of when to replace an asset that produces a physical output, $y(A)$, where $A$ is the state variable representing the age of the asset. The asset's value also depends on the net price of the output, $P$, and the net cost of replacing the asset, $c$.

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 $A \in [0, A^*]$, where $A^*$ is the optimal replacement age. The boundary conditions are given by the value matching condition:

$$V(0) = V(A^*) + c$$

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 an asset which is older than $A^*$ should always be immediately replaced. Once past the age of $A^*$, therefore, the value function is constant: $V(A) = V(A^*) = V(0) - c$, for $A \geq A^*$. No optimality condition is imposed at the lower boundary ($A = 0$) because this boundary is not a decision variable.

Before leaving the example, a potentially misleading interpretation should be discussed. Although it is not unusual to refer to $V(A)$ as the value of an age $A$ asset, this is not quite correct. In fact, $V(A)$ represents the value of the current asset, together with the right to earn returns from future replacement assets. The current asset will be replaced at age $A^*$ and has value equal to the discounted stream of returns it generates:

$$\int_0^{A^*-A} e^{-\rho t} Py(A+t)dt,$$

but the value function is

$$V(A) = \int_0^{A^*-A} e^{-\rho t} Py(A+t)dt + e^{-\rho(A^*-A)}V(A^*)$$

Thus the current asset at age $A$ has value

$$V(A) - e^{-\rho(A^*-A)}V(A^*).$$

**Example: Timber Harvesting**
The previous example 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

$$dS = \alpha(m - S)dt + \sigma\sqrt{S}dz.$$

The state variable here represents the biomass of the stand and 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 $\rho$. The Bellman equation is

$$\rho V = \alpha(m - S)V'(S) + \tfrac{1}{2}\sigma^2 SV''(S),$$

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 $V'(S) = P$.

As in the previous example, the value function refers not to the value of the timber on the stand but rather to the right to cut the timber on the land in perpetuity.

### 10.3.2 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 derivative (the latter is sometimes referred to as a super-contact condition).

**Example: Capacity Choice**
A firm can install capital, $K$, to produce an output with a net return of $P$. Capital produces $Q(K)$ units of output per unit of time, but the capital depreciates at rate $\delta$. The firm wants to determine

$$V(K_t) = \max_{I_\tau} \int_t^\infty e^{-\rho\tau} \Big[ Pq(K_\tau) - CI_\tau \Big] d\tau,$$

s.t.

$$dK = (I - \delta K)dt,$$

together with the constraint that $I \geq 0$. This is an infinite horizon, deterministic control problem. The Bellman's Equation for this problem is

$$\rho V(K) = \max_I Pq(K) - CI - (I - \delta K)V'(K).$$

The KKT condition associated with optimal $I$ is

$$C - V'(K) \geq 0,\ I \geq 0 \text{ and } (V'(K) - C)I = 0.$$

This suggests that the rate of investment should be 0 when the marginal value of capital is less than $C$ and that the rate should be sufficiently high (infinite) to ensure that the marginal value of capital never falls below $C$.

We assume that capital exhibits positive but declining marginal productivity. The optimal control is specified by a value $K^*$ such that investment is 0 when $K > K^*$ (implying low marginal value of capital) and is sufficient high to ensure that $K$ does not fall below $K^*$. If $K$ starts below $K^*$, the investment policy will be to invest at an infinite rate so as to move instantly to $K^*$, incurring a cost of $(K^* - K)C$ in the process. If $K$ starts at $K^*$, the investment rate should be just sufficient to counteract the effect of depreciation. Summarizing, the optimal investment policy is

$$I = \begin{cases} \infty & \text{for } K < K^* \\ \delta K & \text{for } K = K^* \\ 0 & \text{for } K > K^* \end{cases} .$$

The value function for any arbitrary value of $K^*$ can now be determined. To do so, first define the function $T(K, K^*)$ to represent the time it takes for the capital stock to reach $K^*$ is it is current equal to $K$. Clearly $T = 0$ if $K \leq K^*$. In the absence of investment, the capital stock evolves according to $K_{t+h} = e^{-\delta h} K_t$; hence $T = \ln(K/K^*)/\delta$ for $K > K^*$. The value function is

$$V(K; K^*) = \begin{cases} \frac{1}{\rho}\Big(Pq(K^*) - \delta K^* C\Big) - (K^* - K)C & \text{for } K < K^* \\ \int_0^{T(K,K^*)} e^{-\rho\tau} Pq(e^{-\delta\tau} K) d\tau + e^{-\rho T} V(K^*) & \text{for } K > K^* \end{cases} . \quad (14)$$

It is clear that value matching holds; i.e. the value function is continuous. Not so obvious is the fact that the derivative of the value function is also continuous. The left hand derivative is clearly equal to $C$ and a few simple computations will show that the right hand derivative is also.[14] This illustrates the point made above that the usual smooth-pasting condition (i.e., the continuity of the first derivative) is not an optimality condition in barrier control problems.

To determine the optimal choice of $K^*$, notice that we can use the Bellman equation to express the value function for $K > K^*$:

$$V(K; K^*) = \frac{1}{\rho} \left[ Pq(K) - \delta K V_K(K; K^*) \right].$$

[14] For $K > K^*$

$$V'(K) = \int_0^T e^{-\rho\tau} Pq'(e^{-\delta\tau} K) d\tau + \left[ e^{-\rho T} Pq(e^{-\delta T} K) - \rho e^{-\rho T} V(K^*) \right] \frac{dT}{dK}.$$

Using $dT/dK = 1/(\delta K)$ and $T(K^*, K^*) = 0$ and simplifying yields the desired result.

Setting the derivative of this expression with respect to $K^*$ to 0

$$V_{K^*}(K;K^*) = \frac{\delta K}{\rho} V_{KK^*}(K,K^*),$$

and noting that $V_K(K^*,K^*) = C$ implies that $V_{KK^*}(K^*,K^*) = -V_{KK}(K^*,K^*)$, demonstrates that the appropriate optimality condition is to find $K^*$ such that

$$V_{KK}(K^*) = 0,$$

implying continuity of the second derivative.

To complete the problem we apply the Envelope Theorem to the Bellman equation to note that

$$\rho V_K = Pq'(K) - \delta V_K - \delta K V_{KK}.$$

Using the fact that $V_K(K^*) = C$ and $V_{KK}(K^*) = 0$ yields the condition satisfied by the optimal $K^*$:

$$C = \frac{P}{\rho + \delta} q'(K^*).$$

This is the same expression one obtains by setting the derivatives in (14) equal to 0 and solving for $K = K^*$.[15]

### 10.3.3 Discrete State/Control Problems

We turn now to problems involving transitional boundaries. In such problems, controls are not exerted on a continuous state variable to force it to remain within some region. Instead, the boundary typically represents the values of the state at which a decision is made to change from one discrete state to another. In the simplest form of these problems, a termination decision is taken. So-called optional stopping problems include the exercise of American style options and asset abandonment. More complicated problems arise when projects can be activated and deactivated. The problem

[15] We should express a note of caution here. When the depreciation rate is 0, the second derivative condition does not apply and, in fact, the second derivative is discontinuous at the optimal $K^*$. The intuition is clear, however. When $\delta = 0$ the capital stock stays the same unless actively moved. Since disinvestment is not allowed ($I \geq 0$), if the marginal value of capital is less than the cost of capital, no change in the capital stock is warranted. However, if the marginal value of capital is less than $C$, then the capital stock should be immediately increased to the point where the present value of the marginal unit of capital equals the cost of capital:

$$C = Pq'(K^*)/\rho.$$

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 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).

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, net of switching costs, 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).

### 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

$$\rho V(S) = f(S) + \mu(S)V_S(S) + \tfrac{1}{2}\sigma^2(S)V_{SS}(S) \tag{15}$$

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 $S^\tau$. At this point the value function, to be continuous, must equal the reward

$$V(S^a) = R(S^a),$$

(the *value-matching* condition). The optimal choice of $S^a$ is determined by the *smooth pasting* condition

$$V_S(S^*) = R'(S^*);$$

the optimal choice 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 regardless 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. $P$ 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 10.2.2):

$$rV = V_t + (rP - \delta_P) V_P + \tfrac{1}{2}\sigma^2(P)V_{PP}$$

on the continuation region, where $\delta$ 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 included in the Bellman's Equation. The solution involves determining the optimal exercise boundary, $P^*(t)$. Unlike the previous problem, in which the optimal stopping boundary was a single point, the boundary here is a function of time. 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

$$\begin{aligned} V(P,T) &= \max(K - P, 0) && \text{(terminal condition)} \\ V(P^*,t) &= K - P^* && \text{(value matching)} \\ V_P(P^*,t) &= -1 && \text{(smooth-pasting)} \end{aligned}$$

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 $\rho$.

For the finite time case define $\tau$ 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 an infinitely-lived 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). 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 $P^*(\tau)$.

### General Transitional Boundaries

An optimal stopping problem exhibits complete irreversability; an inactivated (exercised) option cannot be reactivated. Optimal stopping problems can put into a more general framework by viewing them as having a binary state variable representing the active/inactive states. In an optimal stopping problem, the cost of moving from the inactive to the active state is effectively infinite, thus precluding this possibility. More generally, however, it may be possible to move back and forth between states.

In general, suppose that there is a state variable, $D$, that can take on integer values from 1 to $n$, as well as a continuous state variable (possibly vector-valued) governed by

$$dS = \mu(S)dt + \sigma(S)dz.$$

The control variable is one that allows any of the $n$ discrete state to be chosen; hence $x(S,D)$ takes on values in the set $\{1,\ldots,n\}$. In addition, there is a cost to switching from $D=i$ to $D=j$ of $F^{ij}$.

A decision rule defines the $n$ sets $\Omega^i$ as the values of $S$ such that $S \in \Omega^i \Rightarrow x(S,i) = i$. $\Omega^i$ is the set of values of $S$ for which a decision rule specifies that the

discrete state remain unchanged. The boundaries of the $\Omega^i$ represent states at which it is optimal to change the value of $D$. It is important to note, however, that the presence of fixed transition costs makes it possible that the $\Omega^i$ are overlapping sets ($\Omega^i \cap \Omega^j \neq \emptyset$).

The solution can be expressed as a set of $n$ functions, $V^i(S)$, representing the value function for each of the $n$ values of the discrete state. From a computational point of view, it is only necessary to define $V^i(S)$ over the points in $\Omega^i$, and to determine the set of switching points $S^{ij}$, for each $(i, j)$. The Bellman equation is

$$rV^i(S) = f(S, i) + \mu(S)V^i_S(S) + \tfrac{1}{2}\sigma^2(S)V^i_{SS}(S),$$

for $S \in \Omega^i$. In addition, at any points, $S^{ij}$ on the boundary of $\Omega^i$ at which a switch from $i$ to $j$ is effected, it must be true that

$$V^i(S^{ij}) = V^j(S^{ij}, j) - F^{ij}$$

to ensure that no arbitrage opportunities exist. Furthermore, if the switch point is optimal, the smooth pasting condition also holds

$$V^i_S(S^{ij}) = V^j_S(S^{ij}).$$

**Example: Entry/Exit**

Consider a firm that 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. In addition to the binary state $\delta$ ($\delta = 0$ for inactive, $\delta = 1$ for active), there is also an exogenous stochastic state representing 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

$$\begin{aligned} \rho V^a &= P - c + \mu(P)V^a_P + \sigma^2(P)V^a_{PP} \\ \rho V^i &= \mu(P)V^i_P + \sigma^2(P)V^i_{PP} \end{aligned} . \tag{16}$$

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

$$V^i_P(P_h) = V^a_P(P_h).$$

In this problem, the exit is irreversible in the sense that reentry is as expensive as initial investment. A refinement of this approach is to allow for temporary suspension of production, with a per unit time maintenance change. Temporary suspension is generally preferable to complete exit (so long as the maintenance charge is not prohibitive). The discrete state for this problem takes on 3 values; its solution is left as an exercise.

### 10.3.4 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 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

$$x = \begin{cases} x_a & \text{if } f_1(S^*) + g_1(S^*)V_S(S^*) > 0 \\ x_b & \text{if } f_1(S^*) + g_1(S^*)V_S(S^*) < 0 \end{cases}$$

This suggests that there is a point, $S^*$, at which

$$f_1(S^*) + g_1(S^*)V_S(S^*) = 0. \tag{17}$$

Assuming that $V_S$ is decreasing in $S$, this suggests that we must solve for two functions, one for $S < S^*$ that solves

$$\rho V^a = f_0(S) + f_1(S)x_a + [g_0(S) + g_1(S)x_a]V_S^a + \tfrac{1}{2}\sigma^2(S)V_{SS}^a \tag{18}$$

and the other for $S > S^*$ that solves

$$\rho V^b = f_0(S) + f_1(S)x_b + [g_0(S) + g_1(S)x_b]V_S^b + \tfrac{1}{2}\sigma^2(S)V_{SS}^b. \tag{19}$$

We will need three side conditions at $S^*$ to completely specify the problem and to find the optimal location of $S^*$, namely that

$$V^a(S^*) = V^b(S^*)$$
$$V_S^a(S^*) = V_S^b(S^*)$$
$$f_1(S^*) + g_1(S^*)V_S(S^*) = 0.$$

Combining these conditions with (18) and (19) we see that

$$V_{SS}^a(S^*) = V_{SS}^b(S^*),$$

i.e., with the optimal choice of $S^*$ the value function is continuous up to its second derivative.

**Example: Harvesting a Renewable Resource**

Consider a manager of a renewable biological 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 $\infty$ 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 $S^C = (S^*, \infty)$.

**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. Consider the case of a firm that has developed a new production technique. Initially production costs are relatively high but decrease as the firm gains more experience with the process. It therefore has an incentive to produce more than it otherwise might due to the future cost reductions it thereby achieves.

To make this concrete, suppose 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. The problem facing the firm is to determine the production rule that maximizes the present value of returns (price less cost times output) over an infinite horizon:

$$\max_x \int_0^\infty e^{-rt}\,(P - C(Q))\,x\,dt,$$

where $r$ is the risk-free interest rate and the two state variables are $P$, the output price, and $Q$, the cumulative production to date. The state transition equations are

$$dP = \mu P\,dt + \sigma P\,dz$$

and

$$dQ = x\,dt,$$

where $x$ is the production rate, which is constrained to lie of the interval $[0, x_c]$. The price equation should be interpreted as a risk-neutral process. The cost function is given by

$$C(Q) = \begin{cases} ce^{-\gamma Q} & \text{if } Q < Q_m \\ ce^{-\gamma Q_m} = \bar{c} & \text{if } Q \geq Q_m \end{cases};$$

once $Q \geq Q_m$, the per unit production cost is a constant but for $Q < Q_m$ it declines exponentially.

The Bellman equation for this problem is

$$rV = \max_x (P - C(Q))\, x + xV_Q + \mu P V_P + \tfrac{1}{2}\sigma^2 P^2 V_{PP}$$

s.t. $0 \leq x \leq x_c$. The problem thus is of the stochastic bang-bang variety with the optimality conditions given by:

$$\begin{aligned} P - C(Q) + V_Q < 0 \quad &\Rightarrow \quad x = 0 \\ P - C(Q) + V_Q > 0 \quad &\Rightarrow \quad x = x_c. \end{aligned}$$

Substituting the optimal production rate into the Bellman Equation and rearranging yields the partial differential equation

$$rV(P,Q) = \mu P V_P(P,Q) + \tfrac{1}{2}\sigma^2 P^2 V_{PP} + \max(0, P - C(Q) + V_Q(P,Q))x_c.$$

The boundary conditions for this problem require that

$$V(0,Q) = 0$$

$$V_P(\infty, Q) = x_c/\delta$$

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 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 $x_c/\delta$.

As illustrated in Figure 10.1, the $(Q, P)$ state space for this problem is divided by a curve $P^*(Q)$ that defines a low price region in which the firm is inactive and a high price region in which it is active. Furthermore, for $Q > Q_m$ the location of $P^*(Q)$ is equal to $c$ because, once the marginal cost is at its minimum level, there is nothing to be gained from production when the price is less than the marginal production cost.

For $Q > Q_m$ the problem is simplified by the fact that $V_Q = 0$. Thus $V$ is a function of $P$ alone and the value of the firm satisfies

$$rV(P) = \mu P V_P(P) + \tfrac{1}{2}\sigma^2 P^2 V_{PP} + \max(0, P - C(Q))x_c.$$

For $Q < Q_m$, on the other hand, $V_Q \neq 0$ and the location of the boundary $P^*(Q)$ must be determined simultaneously with the value function.

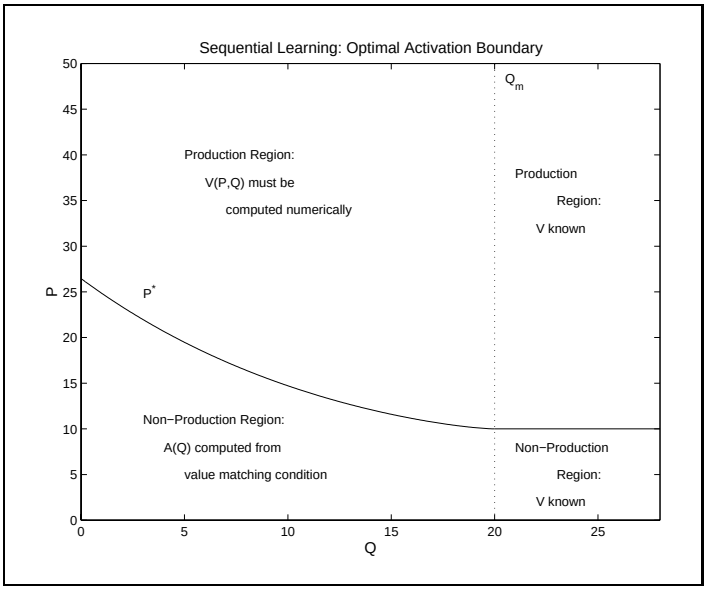


Figure 10.1

A third boundary condition

$$V(P, Q_m) = \bar{V}(P) \text{ (defined below)}$$

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 $Q > Q_m$:

$$\bar{V}(P) = \begin{cases} A_1 P^{\beta_1} & \text{if } P \leq \bar{c} \\ A_2 P^{\beta_2} + \dfrac{P}{\delta} - \dfrac{\bar{c}}{r} & \text{if } P \geq \bar{c}, \end{cases}$$

where the $\beta$ 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 (we know, of course, that $A_1(Q_m)=\bar{c}$)). 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.

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 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

$$V_Q(P,Q) = \begin{cases} -(P^* - C(Q))\left(\frac{P}{P^*}\right)^{\beta_1} & \text{for } P \leq P^* \\ \mathcal{L}V(P,Q) - (P - C(Q)) & \text{for } P \geq P^* \end{cases}$$

**Exercises**

10.1. 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 instantaneous 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 instantaneous interest rate. Assuming that this process is

$$dr = \mu(r,t)dt + \sigma(r,t)dz$$

an extended version of the Feynman-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

$$dr = \kappa(\alpha - r)dt + \sigma dz.$$

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 $B$.

10.2. Pricing Bonds Continued

Given the setting of the previous problem, suppose we take the instantaneous interest rate process to be

$$dr = \kappa(\alpha - r)dt + \sigma\sqrt{r}dz.$$

Verify numerically that $P$ has the form

$$P(r,t,T) = A(t,T)\exp(-B(t,T)r)$$

with

$$A(\tau) = \left( \frac{2\gamma e^{(\gamma+\kappa)\tau/2}}{(\gamma+\kappa)(e^{\gamma\tau}-1)+2\gamma} \right)^{2\kappa\alpha/\sigma^2}$$

and

$$B(\tau) = \frac{2(e^{\gamma\tau}-1)}{(\gamma+\kappa)(e^{\gamma\tau}-1)+2\gamma},$$

where $\gamma = \sqrt{\kappa^2 + 2\sigma^2}$. This can be accomplished by writing a function that returns the proposed value of the bond. This function can be differentiated with respect to $t$ and $r$ to obtain the partial derivatives. The arbitrage condition should be close to 0 for all values of $r$, $t$ and all parameter values.

10.3. Futures Prices

A futures contract maturing in $\tau$ 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 $\delta$ 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 $\delta$ 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.

10.4. Futures Prices Continued

Continuing with the previous question, suppose that the convenience yield is

$$\delta(S,t) = \delta S$$

where $\delta$ 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 $\theta$. Then the futures price solves

$$V_\tau = (r-\delta)SV_S + (\alpha(m-\delta)-\theta)\,V_\delta + \tfrac{1}{2}\sigma^2S^2V_{SS} + \rho\sigma\sigma_\delta SV_{S\delta} + \tfrac{1}{2}\sigma_\delta^2 V_{\delta\delta},$$

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 $B(\tau)$.

10.5. Lookback Options with Geometric Brownian Motion

Suppose the risk neutral process associated with a stock price follows

$$dS = (r-\delta)Sdt + \tfrac{1}{2}\sigma SdW.$$

Show that a lookback strike put option can be written in the form

$$V(S,M,t) = Sv(y,t),$$

where $y = M/S$. Derive the PDE and boundary conditions satisfied by $v$.

10.6. Portfolio Choice with CRRA Utility

For the portfolio choice problem on page 350 show that a utility function of the form $U(C) = (C^{1-\gamma})/(1-\gamma)$ implies an optimal consumption rule of the form $C(W) = cW$. Determine the constant $a$ and, in the process, determine the value function and the optimal investment rule $\alpha(W)$.

10.7. Portfolio Choice with a Risk Free Asset

Suppose that, in addition to $n$ risky assets, there is also a risk-free asset that earns rate of return $r$. The controls for the investor's problem are again $C$, the consumption rate, and the $n$-vector $w$, the fractions of wealth held in the risky assets. The fraction of wealth held in the riskless asset is $1 - \sum_i w_i = 1 - w^\top \underline{1}$.
a) Show that the wealth process can be follows

$$\frac{dW}{W} = [W(r + w^\top(\mu - r\underline{1})) - C]dt + w^\top \sigma dz.$$

b) Write the Bellman's Equation for this problem and the associated first order condition.
c) Show that it is optimal to hold a portfolio consisting of the risk-free asset and a mutual fund with weights proportional to $\Sigma^{-1}(\mu - r\underline{1})$.
d) Derive expressions for $w^\top(\mu - r\underline{1})$ and $w^\top \Sigma w$ and use them to concentrate the Bellman equation with respect to $w$.
e) Suppose that $U(C) = \frac{C^{1-\gamma}}{1-\gamma}$. Verify that the optimal consumption rate is proportional to the wealth level and find the constant of proportionality.

10.8. Portfolio Choice Continued

Continuing the previous problem, define $\lambda(W) = -V'(W)/V''(W)$. Show that $C(W)$ and $\lambda(W)$ satisfy a system of first order differential equations. Use this result to verify that $C$ is affine in $W$ and $\lambda$ is a constant when $U(C) = -e^{-\gamma C}$.

10.9. Stochastic Nonrenewable Resource Management

Suppose that the resource stock discussed on page 354 evolved according to

$$dS = -xdt + \sigma S dz.$$

Verify that the optimal control has the form

$$x(S) = \lambda S^\beta,$$

and, in so doing, determine the values of $\lambda$ and $\beta$. Also obtain an expression for the value function. You should check that your answer in the limiting case that $\sigma = 0$ is the same as that given on page 354.

10.10. Nonrenewable Resources with Stochastic Prices

As in the example on page 354, a resource is extracted at rate $x$, yielding a flow of returns $Ax^{1-\alpha}$. The stock of the resource is governed by $dS = -xdt$. Here, however, we treat $A$ as a random shock process due to randomness in the price of the resource governed by

$$dA = \mu(A)dt + \sigma(A)dz.$$

The firm would like to maximize the expected present value of returns to extraction, using a discount rate of $\rho$.

a) State the firm's optimization problem.
b) State the associated Bellman's equation.
c) State the first order optimality condition and solve for the optimal extraction rate (as a function of the value function and its derivatives).

10.11. Timber Lease

A government timber lease allows a timber company to cut timber for $T$ years on a stand with $B$ units of biomass. The price of cut timber is governed by

$$dp = \alpha(\bar{p} - p)dt + \sigma\sqrt{p}dW.$$

With a cutting rate of $x$ and a cutting cost of $Cx^2/2$, discuss how the company can decide what to pay for the lease, given a current price of $p$ and a discount rate of $\rho$ (assume that the company sells timber as it is cut).

Hint: introduce a remaining stand size, $S$, with $dS = -xdt$ ($S$ is bounded below by 0) and set up the dynamic programming problem.

10.12. Timber Harvesting with Deterministic Growth

Suppose that the timber harvesting problem discussed on page 360 is non-stochastic. The Bellman equation can then be rewritten in the form

$$V' = \frac{\rho}{\alpha}\frac{V}{m - S}.$$

Verify that the solution is of the form

$$V = k(m - S)^{-\rho/\alpha},$$

where $k$ is a constant of integration to be determined by the boundary conditions. There are two unknowns to be determined, $k$ and $S^*$. Solve for $k$ in terms of $S^*$ and derive an optimality condition for $S^*$ as a function of parameters.

10.13. Fishery Management

A monopolist manager of a fishery faces a state transition function

$$dS = [\alpha S(M - S) - x]dt + \sigma^2(S)dW.$$

The price is constant and the cost function has a constant marginal cost that is inversely proportional to the stock level. In addition, a fixed cost of $F$ is incurred if any fishing activity takes place. The reward function can thus be written

$$\left(p - \frac{C}{S}\right) x - F\delta_{x>0}.$$

This is an impulse control problem with two endogenous values of the state, $Q$ and $R$, with $Q < R$. When $S \geq R$, the stock of fish is harvested down to $Q$. Express the Bellman equation for $S \leq R$ and the boundary conditions that determine the location of $Q$ and $R$ (assume a discount rate of $\rho$).

$$\rho V(S) = \alpha S(M - S)V_S(S) + \frac{\sigma^2(S)}{2}V_{SS}(S), \text{ for } S \in [0, R].$$

$$V(R) - V(Q) = p(R - Q) - C\ln(R/Q) - F.$$

$$V_s(R) = p - C/R$$

$$V_s(Q) = p - C/Q.$$

10.14. Capital Investment

Consider an investment situation in which a firm can add to its capital stock, $K$, 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 $\rho$, the Bellman equation for this problem is

$$\rho V(K,P) = \max_I Pq(K) - CI + IV_K(K,P) + \mu P V_P(K,P) + \tfrac{1}{2}\sigma^2 P^2 V_{PP}(K,P).$$

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 capital stock price remains on or above a curve $K^*(P)$. If $K > K^*(P)$, no investment takes place and the value function therefore satisfies

$$\rho V(K,P) = Pq(K) + \mu P V_P(K,P) + \tfrac{1}{2}\sigma^2 P^2 V_{PP}(K,P).$$

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 $\beta_i$ solves $\frac{1}{2}\sigma^2\beta(\beta-1) + \mu\beta - \rho = 0$. It can be shown, for $\rho > \mu > 0$, 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 can drop the subscripts on $A_1(K)$ and $\beta_1$).

At the barrier, the marginal value of capital must just equal the investment cost:

$$V_K(K^*(P),P) = C. \tag{20}$$

Consider now the situation in which the firm finds itself with $K < K^*(P)$ (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 below the

barrier, therefore, is equal to the value at the barrier (for the same $P$) less the cost of the new capital:

$$V(K,P) = V(K^*(P),P) - (K^*(P) - K)C.$$

This suggests that the marginal value of capital equals $C$ when $K < K^*(P)$ and hence does not depend on the current price. Thus, in addition to (20), it must be the case that

$$V_{KP}(K^*(P),P) = 0. \tag{21}$$

Use the barrier conditions (20) and (21) to obtain explicit expressions for the optimal trigger price $P^*(K)$ and the marginal value of capital, $A'(K)$. 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 $(K,P)$.

Write a program to compute and plot the optimal trigger price curve are displayed in using the parameters

$$\begin{aligned} \mu &= 0 \\ \sigma &= 0.2 \\ \rho &= 0.05 \\ c &= 1 \end{aligned}$$

and the following two alternative specifications for $q(K)$:

$$\begin{aligned} q(K) &= \ln(K+1) \\ q(K) &= \sqrt{K}. \end{aligned}$$

10.15. Cash Management

Consider the manager of a cash account subject to random deposits and withdrawals. In the absence 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 $\rho$ 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[16]

$$\rho V(S) = \mu V'(S) + \tfrac{1}{2}\sigma^2 V''(S), \text{ for } S \in [0, S_3]$$

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

$$V'(S_3) = V'(S_2) = (r/\rho - C).$$

[16] 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 $r/\rho$ 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 ($r/\rho$ times the amount of these funds).

The Bellman equation can be solved explicitly:

$$V(S) = A\exp(\alpha S) + B\exp(\beta S),$$

where $\alpha$ and $\beta$ are chosen to solve the differential equation and $A$ and $B$ are chosen to satisfy the side conditions.

Write a MATLAB procedure that accepts the parameters $\mu$, $\sigma$, $\rho$, $r$, $f$, $F$, $c$, and $C$ and returns the parameters $A$, $B$, $\alpha$, $\beta$, $S_1$, $S_2$, and $S_3$. Also determine how the program needs to be modified if the proportional costs ($c$ and $C$) are zero. Check your code 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 $S_3 = 2.2216$.

10.16. Entry/Suspension/Exit

The Entry/Exit problem discussed beginning on page 368 can be extended to allow for temporary suspension of production. Suppose that a maintenance fee of $m$ is needed to keep equipment potentially operative. In the simple entry/exit problem there were two switching costs, $I$ and $E$. Now there are 6 possible switching costs, which will generically be called $F^{ij}$. With $D = 1$ representing the active production state, $D = 2$ the temporarily suspended state and $D = 3$ the exited state, define the Bellman equations and boundary conditions satisfied by the solution.

10.17. Non-Renewable Resource Management

The demand for a nonrenewable resource is given by

$$p = D(q) = q^{-\eta},$$

where $q$ is the extraction rate. For simplicity, assume the resource can be extracted at zero cost. The total stock of the resource is denoted by $S$ (with $S(0) = S_0$), and is governed by the transition function

$$dS = -qdt.$$

a) For the social planner's problem, with the reward function being the social surplus, state the Bellman's equation and the optimality condition, using discount rate $\rho$. Use the optimality condition to find the concentrated Bellman's equation.

b) Guess that $V(S) = \alpha S^{\beta}$. Verify that this is correct and, in doing so, determine $\alpha$ and $\beta$.

c) Determine the time value, $T$, at which the resource is exhausted.

d) Solve the problem using an optimal control (Hamiltonian) approach and verify that the solutions are the same.

10.18. Renewable Resource Management with Adjustment Costs

Consider an extension to the renewable resource problem discussed on page 371. 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.

10.19. Optimal Sales from an Inventory

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 of 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 it is assumed that both the state and the control must be nonnegative. The latter assumes that the agent cannot purchase additional amounts to replenish the inventory, so that sales are irreversible.

The problem can be written as

$$V(S,P,t) = \max_{q(S,P,t)} E_t \int_t^T e^{-rt}\,(qP - kS)\,dt$$

subject to the above constraints.

What is Bellman's equation for this problem? Treat the problem as an optimal stopping problem so $q = 0$ when the price is low and $q = \infty$ when the price is high. At or above the stopping boundary all inventory is sold instantaneously. State the Bellman's equation for the regions above and below the stopping boundary. State the value-matching and smooth-pasting conditions that hold at the boundary.

10.20. Learning-By-Doing with Deterministic Price

Suppose in the sequential learning that the price is deterministic ($\sigma = 0$) and the $r \geq \delta$. In this case, once production is initiated, it is never stopped. Use this to derive an explicit expression for $V(P,Q)$, where $P \geq P^*(Q)$. In this case, because production occurs at all times,

$$V(P,Q) = \int_0^\infty e^{-r\tau}(P_\tau - C(Q_\tau))d\tau,$$

where $P_t$ solves the homogeneous first order differential equation

$$\frac{dP_t}{dt} = (r-\delta)P$$

and

$$\int_0^\infty e^{-r\tau}C(Q_\tau)d\tau = \int_0^{Q_m-Q} e^{-r\tau}C(Q_\tau - Q)d\tau + \bar{c}\int_{Q_m-Q}^\infty e^{-r\tau}d\tau.$$

Also show that, for $P < P^*$, the value function can be written in the form $f(P,P^*(Q))V(P^*(Q),Q)$.

Combining these two results, determine the optimal activation boundary in the deterministic case. Verify that your answer satisfies the Bellman equation.

# Appendix A
# Dynamic Programming and Optimal Control Theory

Many economists are more familiar with optimal control theory than with dynamic programming. This appendix provides a brief discussion of the relationship between the two approaches. 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 deterministic 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

$$\frac{dV_S}{dt} = \rho V_S - f_S - V_S g_S. \tag{22}$$

This can be put in a more familiar form by defining $\lambda = V_S$. Then (22), 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 $\lambda$ the costate variable representing the shadow price of the state variable (expressed in current value terms).[17]

[17] See Kamien and Schwartz, pp. 151-152 for further discussion.

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.

## Appendix B
## 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 $\tau$ 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 every $S$,

$$f'(S^*)\beta(S, S^*) + f(S^*)\left[\frac{\partial\beta(S, S^*)}{\partial S^*} + \frac{\beta(S, S^*)}{1-\beta(0, S^*)}\frac{\partial\beta(0, S^*)}{\partial S^*}\right] = 0. \tag{23}$$

In order to show that this is equivalent to the smooth pasting condition we will use two properties of $\beta$. First, $\beta(S^*, S^*)$ is identically equal to 1, so $\partial\beta(S^*, S^*)/\partial S^* = 0$. Combined with the fact that

$$\left.\frac{dS}{dS^*}\right|_{S=S^*} = 1,$$

this implies

$$\frac{d\beta(S^*, S^*)}{dS^*} = \frac{\partial\beta(S^*, S^*)}{\partial S} + \frac{\partial\beta(S^*, S^*)}{\partial S^*} = 0$$

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 (23) 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 C
## 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 $S^\tau$ as a switching point, e.g., set $x = x_a$ for $S < S^\tau$ and $x = x_b$ for $S > S^\tau$.[18] At $S = S^\tau$ we choose $x$ in such a way that $dS/dt = 0$. Summarizing, define

$$x(S, S^a) = \begin{cases} x_a & \text{if } S < S^a \\ -\dfrac{g_0(S)}{g_1(S)} & \text{if } S = S^a \\ x_b & \text{if } S > S^a \end{cases},$$

with $x_a < -g_0(S^a)/g_1(S^a) < x_b$. The value function satisfies the differential equation

$$V(S, S^a) = \frac{1}{\rho}\Big(f_0(S) + f_1(S)x(S, S^a) + \Big[g_0(S) + g_1(S)x(S, S^a)\Big]V_S(S, S^a)\Big), \quad (24)$$

which, evaluated at $S = S^\tau$, yields

$$V(S^a, S^a) = \frac{1}{\rho}\left(f_0(S^a) - f_1(S^a)\frac{g_0(S^a)}{g_1(S^a)}\right). \quad (25)$$

In spite of the discontinuity of the control at $S^\tau$, the value function is continuous, as is readily apparent by writing it as

$$V(S, S^a) = \int_0^\infty e^{-\rho t}\left(f_0(S) + f_1(S)x(S, S^a)\right)dt,$$

and noting that as $S$ approaches $S^\tau$ from below (above), the amount of time during which the control is set at $x_a$ ($x_b$) goes to 0.

[18] 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.

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:[19]

$$V_S(S^a, S^a) = -\frac{f_1(S^a)}{g_1(S^a)}. \tag{26}$$

So far, however, we have only considered the value function for the control $S^\tau$. To choose the control optimally, we must pick $S^\tau$ to satisfy

$$V_{S^a}(S, S^a) = 0.$$

For $S \neq S^a$ we can differentiate (24) to see that

$$\begin{aligned} V_{S^a}(S, S^a) &= \tfrac{1}{\rho}\Big[f_1(S) + g_1(S)V_S(S, S^a)\Big]x_{S^a}(S, S^a) \\ &\qquad + g_1(S)x(S, S^a)V_{SS^a}(S, S^a). \end{aligned} \tag{27}$$

However, except at $S = S^\tau$, $x_{S^\tau}(S, S^\tau)$ and $V_{SS^a}(S, S^a)$ are zero and hence we only need to set this derivative to zero at $S = S^\tau$. (27) 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

$$\frac{dV(S^a, S^a)}{dS^a} = V_S(S^a, S^a) + V_{S^a}(S^a, S^a).$$

Rearranging this and using (25) and (26) we get

$$\begin{aligned} V_{S^a}(S^a, S^a) &= \frac{dV(S^a, S^a)}{dS^a} - V_S(S^a, S^a) \\ &= \frac{1}{\rho}\frac{d\left(f_0(S^a) - g_0(S^a)\frac{f_1(S^a)}{g_1(S^a)}\right)}{dS^a} + \frac{f_1(S^a)}{g_1(S^a)}. \end{aligned}$$

[19] To determine the limit from below, note that continuity of $V$ implies that

$$\begin{aligned} \lim_{S \nearrow S^a} V(S, S^a) &= \lim_{S \nearrow S^a} \frac{1}{\rho}\left[f_0(S) + f_1(S)x_a + (g_0(S) + g_1(S)x_a)\, V_S(S, S^a)\right] \\ &= \frac{1}{\rho}\left[f_0(S^a) + f_1(S^a)x_a + (g_0(S^a) + g_1(S^a)x_a) \lim_{S \nearrow S^a} V_S(S, S^a)\right] \\ &= \frac{1}{\rho}\left[f_0(S^a) - \frac{f_1(S^a)g_0(S^a)}{g_1(S^a)}\right] \equiv V(S^a, S^a). \end{aligned}$$

Rearranging, we see this expression implies that

$$\begin{aligned} (g_0(S^a) + g_1(S^a)x_a) \lim_{S \nearrow S^a} V(S, S^a) &= -\left(f_1(S^a)x_a + \frac{f_1(S^a)g_0(S^a)}{g_1(S^a)}\right) \\ &= -\frac{f_1(S^a)}{g_1(S^a)}\left(g_0(S^a) + g_1(S^a)x_a\right). \end{aligned}$$

The same exercise can be applied to solving for the limit from above.

Thus the optimal switching points are found by solving for the roots of this expression. It is possible that 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.

## Bibliographic Notes

Arbitrage methods for solving financial asset pricing problems originated with Black and Scholes and Merton. The literature is now vast. A good introductory level discussion in found in Hull. For a more challenging discussion, see Duffie. The mathematical foundations for modern asset pricing theory are discussed at an introductory level in Hull and Neftci. See also Shimko.

Discussions of exotic option pricing models are found in Hull and Wilmott. Goldman et al. contains the original derivation of the boundary condition for lookback options. Affine diffusion models are discussed in Duffie and Kan, Dai and Singleton and Fackler.

Introductory level treatments of stochastic control problems are available in Dixit and Dixit and Pindyck. These books contain numerous examples as well as links to other references. A more rigorous treatment in found in Fleming and Rishel. The renewable resource harvesting problem is from Pindyck, the optimal investment from Cox et al., the portfolio choice example from Merton (1969) and Merton (1971).

Boundary conditions associated with stochastic processes are discussed by Feller, who devised a classification scheme for diffusion processes with singular boundaries (see discussion by Bharucha-Reid, sec. 3.3, and Karlin and Taylor (1981), Chap. 15.).

Kamien and Schwartz is a classic text on solving dynamic optimization problems in economics; its primary focus is on deterministic problems solved via calculus of variations and use of Hamiltonian methods but contains a brief treatment of dynamic programming and control of Ito processes (Chapters 20 and 21). Other useful treatments of deterministic problems are found in Dorfman and Chiang. Malliaris and Brock contains an overview of Ito processes and stochastic control, with numerous examples in economics and finance. Duffie contains a brief introductory treatment of stochastic control, with a detailed discussion of the portfolio choice problem first posed in Merton (1969) and Merton (1971). A standard advanced treatment of stochastic control is Fleming and Rishel.

Free boundary problems are increasingly common in economics. Dixit (1991), Dixit (1993a) and Dixit and Pindyck contain useful discussions of these problems. Several of the examples are discussed in these sources. Dixit (1993b) provides a good introduction to stochastic control, with an emphasis on free boundary problems. Dumas discusses optimality conditions; see also the articles in Lund and Oksendal. Hoffman and Sprekels and Antonsev et al. are proceedings of conferences on free boundary problems with an emphasis on problems arising in physical sciences.

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, Hershleifer. Recently, ?? discussed the

stochastic version of the problem. The entry/exit example originates with Brennan and Schwartz and McDonald and Siegel. Numerous authors have discussed renewable resource management problems; see especially Mangel. The stochastic bang-bang problem is discussed most fully in Ludwig and Ludwig and Varrah, where detailed proofs and a discussion of the multiple equilibria situation can be found. The proof in the appendix to this chapter is modeled after a similar proof in Ludwig. The learning-by-doing example is from Majd and Pindyck and is also discussed in Dixit and Pindyck.

# Chapter 11

# Continuous Time Models: Solution Methods

In the previous chapter we saw how continuous time economic models, whether deterministic or stochastic, result in solution functions that satisfy differential equations. 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 Sections 5.7 and 6.8.3. 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 possible, it is usually faster to use such recursive solution techniques.

Numerous methods have been developed for solving PDEs. We concentrate on a particular approach that encompasses a number of the more common methods and which builds nicely on the material already covered in this book. Specifically, the true but unknown solution will be replaced with a convenient approximating function, the parameters of which will be determined using collocation. For initial value type problems (IVPs), this approach will be combined with a recursive algorithm. We will also discuss free boundary problems. The basic approach for such problems is to solve the model taking the free boundary as given and then use the optimality condition to identify the location of the boundary.

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, simulation methods and methods that discretize control problems and solve the related discrete problem. Although all of these methods have their place, we feel that providing a general framework that works to solve a wide variety of problems and builds on general methods developed in previous chapters is of more value than an encyclopedic account of existing approaches. Much of what is discussed here should look and feel familiar to readers that have persevered up to this point.[1] We do, however, include some references to other approaches in the bibliographical notes at the end of the chapter.

## 11.1 Solving Arbitrage-based Valuation Problems

In the previous chapter it was shown that financial assets often satisfy an arbitrage condition in the form of the PDE

$$r(S)V = \delta(S) + V_t + V_S\mu(S) + \tfrac{1}{2}\text{trace}(\sigma(S)\sigma(S)^\top V_{SS}).$$

The specific asset depends on the boundary conditions. For an asset that has a single payout at time $T$, the only boundary condition is of the form $V(S,T) = R(S)$ and, as there are no dividends, $\delta = 0$. For zero-coupon default-free bonds the boundary condition is $R(S) = 1$. For futures, European call options and European put options written on an underlying asset with price $p = P(S)$, the boundary conditions are, respectively, $R(S) = P(S)$ and $R(S) = \max(0, P(S) - K)$ and $R(S) = \max(0, K - P(S))$, where $K$ is the option's strike price.

Asset pricing problems of this kind are more easily expressed in terms of time-to-maturity rather than calendar time; let $\tau = T - t$. We will work with $V(S,\tau)$ rather than $V(S,t)$, necessitating a change of sign of the time derivative: $V_\tau = -V_t$.

The problem, of course, is that the functional form of $V$ is unknown. Suppose, however, it is approximated with a function of the form $V(S,\tau) \approx \phi(S)c(\tau)$, where $\phi$ is a suitable $n$-dimensional family of approximating functions and $c(\tau)$ is an $n$-vector of time varying coefficients. When the state variable is one-dimensional, the arbitrage condition can be used to form a residual equation of the form[2]

$$\phi(S)c'(\tau) \approx \Big[\mu(S)\phi'(S) + \tfrac{1}{2}\sigma^2(S)\phi''(S) - r(S)\phi(S)\Big]c(\tau) = \beta(S)c(\tau). \qquad (1)$$

[1] It would be useful to at least be familiar with the material in Chapter 6.

[2] For multi-dimensional states the principle is the same but the implementation is a bit messier. We discuss this in Appendix 11.A.

A collocation approach to determining the $c(\tau)$ is to select a set of $n$ values for $S$, $s_i$, and to solve the residual equation with equality at these values. The residual function can then be written in the form

$$\Phi c'(\tau) = Bc(\tau),$$

where $\Phi$ and $B$ are both $n \times n$ matrices. This is a first order system of ordinary differential equations in $\tau$, with the known solution $c(\tau) = \exp(\tau\Phi^{-1}B)c_0$, where $\Phi c_0$ satisfies the boundary condition $R(S)$ evaluated at the $n$ values of the $s_i$ (note: the exponential is a matrix exponential which can be computed using the MATLAB function `expm`).

Before illustrating this approach, a few additional comments are in order. It may be desirable to impose additional boundary conditions. This would require redefining $\Phi$ and $B$ using fewer than $n$ nodes and to concatenate to these any additional equations needed to impose the boundary conditions. Generally, this is not needed if the behavior at the boundaries is regular enough. The issue becomes more critical when free boundary problems are encountered. Another issue is especially important in option pricing problems, for which the terminal boundary is not smooth but rather has a kink at the strike price $K$. This suggests that the use of a polynomial approximation may not be appropriate. Instead a cubic spline, possibly with extra nodes at $S = K$, or a piece-wise linear approximation with finite difference derivatives may be in order.

**Example: Pricing Bonds**
The CIR (Cox-Ingersoll-Ross) bond pricing model assumes that the risk-neutral process for the short interest rate is given by

$$dr = \kappa(\alpha - r)dt + \sigma\sqrt{r}dz.$$

Expressing the value of a bond in terms of time-to-maturity ($\tau$), a bond paying 1 unit of account at maturity, has value $V(r, \tau)$ that solves

$$V_\tau = \kappa(\alpha - r)V_r + \tfrac{1}{2}\sigma^2 r V_{rr} - rV,$$

with initial condition $V(r, 0) = 1$.

To solve this model, first choose a family of approximating functions with basis $\phi(r)$ and $n$ collocation nodes, $r_i$. Letting the basis functions and their first two derivatives at these points be defined as the $n \times n$ matrices $\Phi_0$, $\Phi_1$ and $\Phi_2$, a system of collocation equations is given by

$$\Phi_0 c'(\tau) = [\kappa(\alpha - r)\Phi_1 + \tfrac{1}{2}\sigma^2 r\Phi_2 - r\Phi_0]c(\tau) = Bc(\tau).$$

The term $r\Phi_0$ is an abuse of notation; it indicates multiplying the $n \times 1$ vector $r$ by an $n \times n$ matrix $\Phi_0$. Such a term is more properly written as $\text{diag}(r)\Phi_0$ and can be obtained in MATLAB using element-by-element multiplication as

```
r(:,ones(n,1)).*Phi0
```

(see code below). The same comments also apply to the first and second order terms. The following function solves the CIR bond pricing problem.

```
function c=cirbond(fspace,tau,kappa,alpha,sigma)
  % Define nodes and basis
  r=funnode(fspace);
  Phi0=funbas(fspace,r,0);
  Phi1=funbas(fspace,r,1);
  Phi2=funbas(fspace,r,2);
  % Evaluate parameters
  m=kappa*(alpha-r);
  s=0.5*sigma.^2*r;
  % Define and solve the linear differential equation in the coefficients
  u=ones(size(r,1),1);
  B=m(:,u).*Phi1+s(:,u).*Phi2-r(:,u).*Phi0;
  B=Phi0\B;
  c0=Phi0\u;
  c=expm(full(tau*B))*c0;
```

The function's input arguments include a function definition structure `fspace` indicating the family of approximating functions desired, the time to maturity, $\tau$, and the model parameters, $\kappa$, $\alpha$ and $\sigma$. The function returns the coefficient vector $c_\tau$. A script file, `demfin01.m`, demonstrates the use of the procedure. It uses a Chebyshev polynomial approximation of degree $n = 20$ on the interval $[0, 2]$. The solution function for a 30-year bond with parameter values $\kappa = 0.1$, $\alpha = 0.05$ and $\sigma = 0.1$ is plotted in Figure 11.1.

Two points are in order concerning the procedure. First, MATLAB's matrix exponential function `expm` requires that its argument be a full rather than a sparse matrix. Some basis functions (e.g., spline and piecewise linear) are stored as sparse matrices, so this ensures that the code will work regardless of the family of functions used.

Second, the procedure uses the standard nodes for a given approximating family of functions. This typically requires upper and lower bounds to be specified. For the process used in the bond pricing example, a natural lower bound of 0 can be used. The upper bound is trickier, because the natural upper bound is $\infty$. Knowledge of the underlying nature of the problem, however, should suggest an upper bound for the rate of interest. We have used 2, which should more than suffice for countries that are not experiencing hyper-inflation.

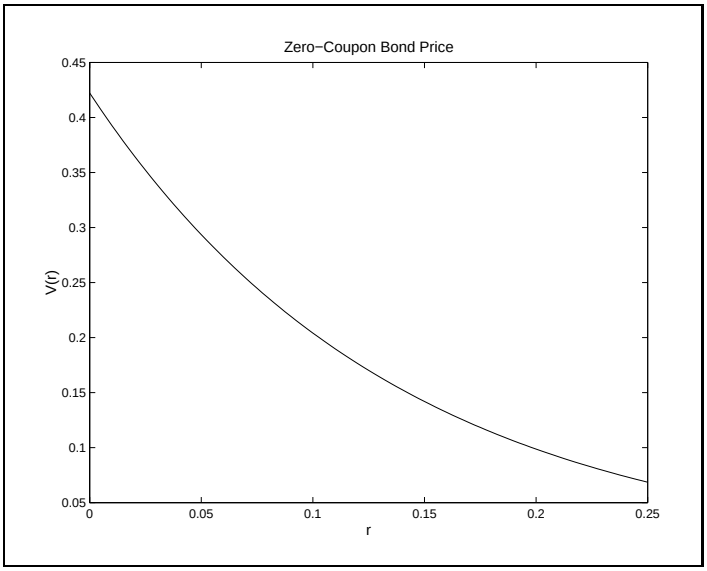


Figure 11.1

More generally, one should use an upper bound that ensures the result is not sensitive to the choice in regions of the state space that are important. In practice, this may necessitate some experimentation. A useful rule of thumb is that the computed value of $V(S)$ is not sensitive to the choice of $\bar{S}$ if the probability that $S_T = \bar{S}$, given $S_t = S$, is negligible. For infinite horizon problems with steady state probability distributions, one would like the steady state probability of $\bar{S}$ to be negligible.

For this example, a known solution to the bond pricing problem exists (see exercise 10.2 on p. 376). The closed form solution can be used to compute the approximation error function, which is shown in Figure 11.2. The example uses a Chebyshev polynomial basis of degree $n = 20$; it is evident that this is more than sufficient to obtain a high degree of accuracy.

### 11.1.1 Extensions and Refinements

The approach to solving the asset pricing problems just described replaces the original arbitrage condition with one of the form

$$\Phi c'(\tau) = Bc(\tau),$$

with $\Phi c(0) = V_0$. The known solution $c(\tau) = \exp(\tau\Phi^{-1}B)\Phi^{-1}V_0$ can be put in recursive form

$$c(\tau + \Delta) = \exp(\Delta\Phi^{-1}B)c(\tau) = Ac(\tau).$$

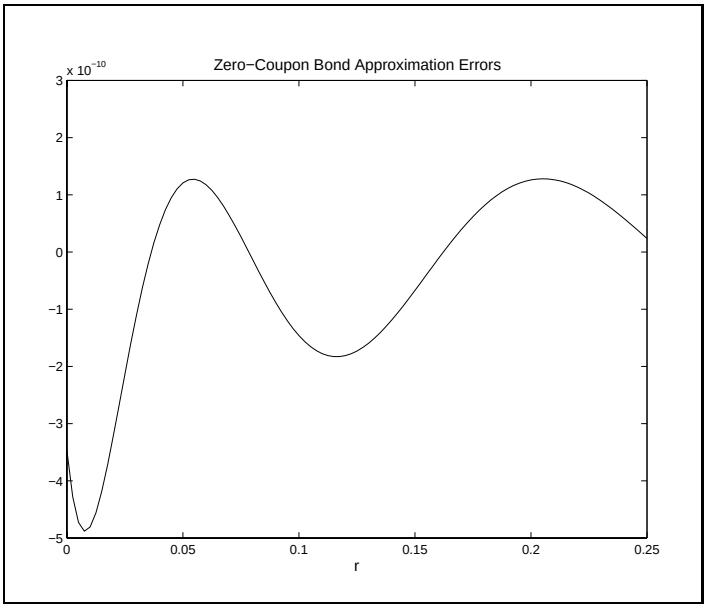


Figure 11.2

The $n \times n$ matrix $A$ need only be computed once and the recursive relationship may then be used to compute solution values for a whole grid of evenly spaced values of $\tau$.

In the approach taken above, the existence of a known solution to the collocation differential equation is due to the linearity of the arbitrage condition in $V$ and its partial derivatives. If linearity does not hold, we will still be able to express the system in the form $\Phi c'(t) = B(c(t))$, which can be solved using any convenient initial value solver such as the Runge-Kutta algorithm described in Section 5.7 or any of the suite of MATLAB ODE solvers. This approach has been called the extended method of lines. The name comes from a technique called the method of lines, which treats $\Phi = \mathrm{I}_n$ and uses finite difference approximations for the first and second derivatives in $S$. The values contained in the $c(t)$ vector are then simply the $n$ values of $V(s_i, t)$. The extended method of lines simply extends this approach by allowing for arbitrary basis functions.

We should point out that the system of ODEs in the extended method of lines is often "stiff". This is a term that is difficult to define precisely and a full discussion is beyond the scope of this book. Suffice it to say, a stiff ODE is one that operates on very different time scales. The practical import of this is that ordinary evolutionary solvers such as Runge-Kutta and its refinements must take very small time steps to solve stiff problems. Fortunately, so-called implicit methods for solving stiff problems

do exist. The MATLAB ODE suite provides two stiff solvers, `ode15s` and `ode23s`.

It is also possible to use finite difference approximations for $\tau$; indeed, this is perhaps the most common approach to solving PDEs for financial assets. Expressed in terms of time-to-maturity ($\tau$), a first order approximation with a forward difference (in $\tau$) transforms (1) to

$$\phi(S)\frac{c(\tau+\Delta)-c(\tau)}{\Delta} = \beta(S)c(\tau),$$

or, equivalently,

$$\phi(S)c(\tau+\Delta) = [\phi(S)+\Delta\beta(S)]c(\tau).$$

Expressing this in terms of basis matrices evaluated at $n$ values of $S$ leads to

$$c(\tau+\Delta) = [I_n + \Delta\Phi^{-1}B]c(\tau).$$

This provides an evolutionary rule for updating $c(\tau)$, given the initial values $c(0)$. $[I_n + \Delta\Phi^{-1}B]$ is a first order Taylor approximation (in $\Delta$) of $\exp(\Delta\Phi^{-1}B)$. Hence the first order differencing approach leads to errors of $O(\Delta^2)$.

A backwards (in $\tau$) differencing scheme can also be used

$$\phi(S)\frac{c(\tau)-c(\tau-\Delta)}{\Delta} = \beta(S)c(\tau),$$

leading to

$$c(\tau-\Delta) = [I_n - \Delta\Phi^{-1}B]c(\tau)$$

or

$$c(\tau+\Delta) = [I_n - \Delta\Phi^{-1}B]^{-1}c(\tau).$$

$[I_n - \Delta\Phi^{-1}B]^{-1}$ is also a first order Taylor approximation (in $\Delta$) of $\exp(\Delta\Phi^{-1}B)$ so this method also has errors of $O(\Delta^2)$.

Although it may seem like the forward and backwards approaches are essentially the same, there are two significant differences. First, the backwards approach defines $c(\tau)$ implicitly and the update requires a linear solve using the matrix $[I_n - \Delta\Phi^{-1}B]$. The forward approach is explicit and requires no linear solve. This point is relatively unimportant when the coefficients of the differential equation are constant in $\tau$ because the inversion would need to be carried out only once. The point becomes more significant when the coefficients are time varying. As we shall see, this can happen even when the state process has constant (in time) coefficients, especially in free boundary problems.

The need for a linear solve would seem to make the backward (implicit) approach less desirable. It is possible, however, that the explicit forward approach is unstable. Both approaches replace the differential system of equations with a system of difference equations of the form

$$x_{\tau+\Delta} = Ax_\tau.$$

It is well known that such a system is explosive if any of the eigenvalues of $A$ are greater than 1 in absolute value.

In applications of the kind found in financial applications, the matrix $A = [I_n + \Delta\Phi^{-1}B]$ can be assured of having small eigenvalues only by making $\Delta$ small enough. On the other hand, the implicit method leads to a difference equation for which $A = [I_n - \Delta\Phi^{-1}B]^{-1}$, which can be shown to be stable for any $\Delta$. Practically speaking, this means that the explicit may not be faster than the implicit method and may produce garbage if $\Delta$ is not chosen properly. If the matrix $A$ is explosive, small errors in the approximation will be magnified as the recursion progresses, causing the computed solution to bear no resemblance to the the true solution.

## A Matlab Asset Pricing Function: FINSOLVE

Due to the common structure of the arbitrage pricing equation across a large class of financial assets, it is possible to write general procedures for asset pricing. Such a procedure, `finsolve`, for solving an arbitrage condition for an asset that pays no dividends is shown in below. The function requires five inputs, `model`, `fspace`, `alg`, `snodes` and `N`. The first input, `model`, is a structure variable with the following fields:

| | |
|---|---|
| `func` | the name of the problem definition file |
| `T` | the time to maturity of the asset |
| `params` | a cell array of additional parameters to be passed to `model.func` |

A template for the function definition file is:

```
out1=func(flag,S,t,additional parameters);
  switch flag
   case 'rho'
     out1 = instantaneous risk-free interest rate
   case 'mu'
     out1 = drift on the state process
   case 'sigma'
     out1 = volatility on the state process
   case 'V0'
     out1 = exercise value of the asset
   end
```

The function uses the modified method of lines by default if `alg` is unspecified but explicit (forward) or implicit (backward) finite differences can also be used to represent the derivative in $\tau$ by specifying the `alg` argument to be either `'implicit'` or `'explicit'` (the default is `'lines'`). In addition, a method known as the Crank-Nicholson method, which averages the implicit and explicit methods, can be obtained by specifying `'CN'`.

```
function [c,V,A]=finsolve(model,fspace,alg,s,N)

if ~exist('alg','var') | isempty(alg), alg='lines'; end
if ~exist('N','var') | isempty(N), N=1; end

probfile=model.func;
T=model.T;
n=prod(fspace.n);

% Compute collocation matrix
mu=feval(probfile,'mu',s,[],model.params{:});
sigma=feval(probfile,'sigma',S,[],model.params{:});
rho=feval(probfile,'rho',S,[],model.params{:});
V0=feval(probfile,'V0',S,[],model.params{:});

n=fspace.n;
Phi0=funbas(fspace,s,0);   % compute basis matrices
Phi1=funbas(fspace,s,1); Phi2=funbas(fspace,s,2);
v=0.5*sigma.*sigma; u=ones(n,1);
B=mu(:,u).*Phi1+v(:,u).*Phi2-rho(:,u).*Phi0;
B=funfitxy(fspace,Phi0,B);
c0=funfitxy(fspace,Phi0,V0);

Delta=T/N;

switch method
case 'lines'
  A=expm(full(Delta*B));
case 'explicit'
  A=eye(n)+Delta*B;
case 'implicit'
  A=inv(eye(n)-Delta*B);
case 'CN'
```

```
  A=(inv(eye(n)-Delta*B) + eye(n)+Delta*B)/2;
otherwise
  error('Method option is invalid')
end

c=zeros(n,N+1);
c(:,1)=c0;
for i=2:N+1
  c(:,i)=A*c(:,i-1);
end
```

The next section uses this function to solve the Black-Scholes option pricing model (the MATLAB file `demfin01` optionally uses `finsolve` to solve the CIR bond pricing example).

**Example: Black-Scholes Option Pricing Formula**
In Section 10.1, the Black-Scholes option pricing formula was introduced. The assumption underlying this formula is that the price of a dividend protected stock has risk-neutral dynamics given by

$$dS = rSdt + \sigma Sdz.$$

The arbitrage condition is

$$V_\tau = rSV_S + \tfrac{1}{2}\sigma^2 S^2 V_{SS} - rV$$

with the initial condition $V(S,0) = \max(S-K,0)$.

A script file using the `finsolve` function is given below (see `demfin02`) . The family of piece-wise linear functions with finite difference approximations for the derivatives (with prefix `lin`) is used. 50 evenly spaced nodal values on $[0, 2K]$ are used, along with 75 time steps using the implicit (backward in $\tau$) method. Notice that the function `finsolve` returns an $n \times N+1$ matrix of coefficients; the first column contains coefficients that approximate the terminal value. If only the values for time-to-maturity $T$ are desired, all but the last column of the output can be discarded (hence the `c=c(:,end);` line). The approximation can be evaluated at arbitrary values of $S$ using `funeval(c,fspace,S)`. The delta and gamma of the option can also be evaluated using `funeval`.

```
% Define parameters
r=.05; delta=0; sigma=0.2; K=1; T=1; put=0;
```

```
clear model
model.func='pfin02';
model.T=T;
model.params={r,delta,sigma,K,put};

n=50;
fspace=fundefn('lin',n,log(K/3),log(3*K));
s=funnode(fspace);

% Call solution algorithm
c=finsolve(model,fspace,'implicit',s,75);
c=c(:,end);
```

The problem definition file for this example follows:

```
function out1=pfin02(flag,S,t,r,delta,sigma,K,put);
n=size(S,1);
switch flag
case 'rho'
  out1=r+zeros(n,1);
case 'mu'
  out1= (r-delta-0.5*sigma.^2);
case 'sigma'
  out1=sigma;
case 'V0'
  if put
    out1=max(0,K-exp(S));
  else
    out1=max(0,exp(S)-K);
  end
end
```

As a closed form solution exists for this problem, we can plot the approximation errors produced the method (the procedure `BlackSch` is available in the `CompEcon` toolbox). These are shown in Figure 11.3. The maximum absolute error is $5.44 \times 10^{-4}$. It is simple to experiment with the alternate methods by changing the `alg` argument to the `finsolve` function. The family of approximating functions can also be changed easily by changing the input to the `fundef` function. The approximation errors in Table 11.1 were obtained in this manner.

The approximation errors for all of these methods are roughly equivalent, with a slight preference for the cubic spline. Note, however, that the explicit approach

**Table 11.1: Option Pricing Approximation Errors**

| | Method | | |
|---|---|---|---|
| Function Family | Lines | Implicit | Explicit |
| Piecewise-linear | 4.85 | 5.44 | 5.31 |
| Cubic Splines | 2.89 | 2.31 | 3.20 |

Maximum absolute errors on $[0, 2K]$ times $10^{-4}$.
Explicit cubic spline uses 250 time steps; other methods use 50 steps.

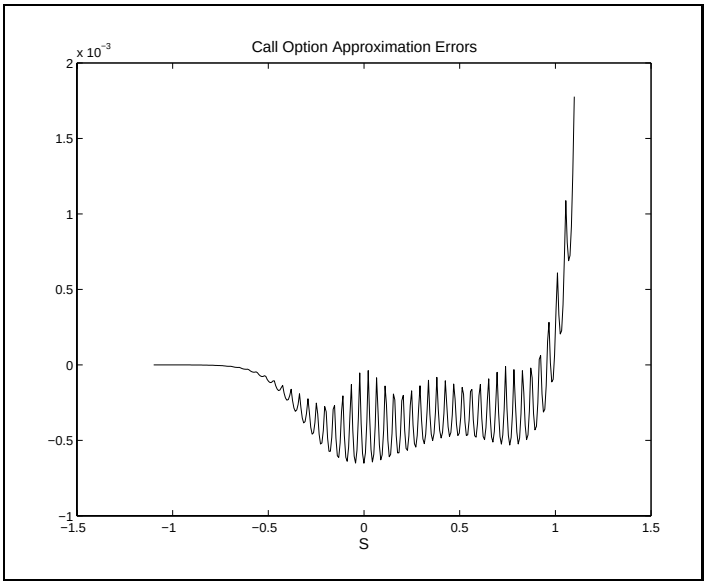


Figure 11.3

using the cubic spline basis is explosive with under 200 time steps and the table gives approximation errors using 250 time steps. It should also be noted that a polynomial approximation does a very poor job in this problem due to its inability to adequately represent the initial condition, which has a discontinuity at $K$ in its first derivative, and, more generally, because of the high degree of curvature near $S = K$.

## 11.2 Solving Stochastic Control Problems

In the previous chapter we saw that for problems of the form

$$V(S) = \max_{x(S)} \int_t^\infty e^{-\rho\tau} f(S, x) d\tau, \text{ s.t. } dS = g(S, x)dt + \sigma(S)dz,$$

Bellman's equation takes the form

$$\rho V(S) = \max_{x(S)} f(S,x) + g(S,x)V'(S) + \tfrac{1}{2}\sigma^2(S)V''(S),$$

possibly subject to boundary conditions.

When the functional form of the solution is unknown, the basic strategy for solving such problems will be essentially the same as in the discrete time case. The value function will be approximated using $V(S) \approx \phi(S)c$, where $c$ is an $n$-vector of coefficients. For infinite horizon problems, $c$ can be found using either a value function and a policy function iteration.

Given a guess of the value of $c$, the optimal value of the control can be solved (in principle) for a set of nodal values of $S$.

$$\rho\phi(s_i)c = \max_{x_i} f(s_i,x_i) + g(s_i,x_i)\phi'(s_i)c + \tfrac{1}{2}\sigma^2(s_i)\phi''(s_i)c.$$

This leads to the first order conditions

$$f_x(s_i,x_i) + g_x(s_i,x_i)\phi'(s_i)c = 0,$$

which may admit a closed form solution, $x_i = x^*(s_i;c)$.

If there are no relevant boundary conditions, $n$ values of $S$ are used to form the $n \times n$ basis matrices $\Phi_0$, $\Phi_1$ and $\Phi_2$. The three vectors defined by $f_i = f(s_i,x_i)$, $m_i = g(s_i,x_i)$ and $v_i = 0.5\sigma^2(s_i)$ are also computed. A function iteration algorithm can then be expressed as

$$c \leftarrow \frac{1}{\rho}\Phi_0^{-1}\Big(f + [m\Phi_1 + v\Phi_2]\,c\Big)$$

(as noted earlier, terms like $m\Phi_1$ are an abuse of notation and signify $\text{diag}(m)\Phi_1$). Policy function iteration uses

$$c = [\rho\Phi_0 - m\Phi_1 - v\Phi_2]^{-1} f.$$

If there are relevant boundary conditions, the number of nodal values of $S$ can be less than $n$ by the number of additional conditions. Generally boundary conditions are linear in the value function and its derivatives and hence are linear in the approximation coefficients. These conditions can, therefore, be appended to the collocation conditions from the Bellman's Equation and $c$ can be updated in essentially the same manner. This approach will be used extensively for solving free boundary in the next section.

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.

A general solver routine for stochastic control problems with continuous control variables can be developed along the same lines as the solver for the discrete time case (discussed beginning on p. ??). The method starts with an initial guess of the coefficients of an approximation of the value function. It then computes the optimal control given this guess. The value function coefficients are then updated using either policy or function iteration. This iterative process occurs until a convergence criterion is met.

There are, in fact, some distinct computational advantages to using continuous time models because of three related facts. First, the Bellman equation is not stochastic; there is no need to perform numerical integration to compute an expectation. Second, to implement a Newton solution to the optimization problem, the first order condition, $f_x(s,x) + V_s(s)g_x(s,x) = 0$, and its derivative need only be evaluated at nodal values of the state. Third, the first order condition is relatively simple and can often be solved explicitly in the form $x^* = x(s, V_s)$, thereby eliminating entirely the need to perform use numerical optimization methods to determine the conditional optimal control.

Even if no explicit solution is available, the computation of second derivative of the conditional value function is easier.[3] In the discrete time case

$$v_{xx} = f_{xx} + V_s g_{xx} + {g_x}^\top V_{ss} g_x.$$

In the continuous time case, the last of these terms does not appear because the value of the state at which $V$ is evaluated is independent of $x$.

A general solver for continuous time stochastic control problems is provided by `scsolve`. This solver requires that the user specify a model structure and problem definition file as well as a family of approximating a functions, a set of collocation nodes and initial values for the value function at the collocation nodes. Optionally, an initial value for the optimal control may be passed. The model structure should

[3] If the variance term $\sigma$ is a function of $x$, the computation of the first and second derivatives must account for this.

$$v_x = f_x + V_s g_x + \mathrm{vec}(V_{ss}\sigma)^\top \sigma_x$$

and

$$v_{xx} = f_{xx} + V_s g_{xx} + {\sigma_x}^\top (I_d \otimes V_{ss})\sigma_x + \mathrm{vec}(V_{ss}\sigma)^\top \sigma_{xx}.$$

Here $\sigma_x$ is $d^2 \times p$ and $\sigma_x$ is $d^2 \times p \times p$. The matrix product $\mathrm{vec}(V_{ss}\sigma)^\top \sigma_{xx}$ multiplies a $1 \times d^2$ matrix by a $d^2 \times p \times p$ array and returns a $p \times p$ array. In MATLAB this type of matrix multiplication is accomplished by

`reshape(A*reshape(B,d*d,p*p),p,p)`.

contain the following fields:

| | |
|---|---|
| `func` | the name of the problem definition file |
| `params` | a cell array of additional parameters to be passed to `model.func` |

For many continuous time problems, a solution to the first order conditions (as a function of $S$ and $V_S$) can be obtained in closed form. If this is the case, the problem definition file should return these values when a `flag` value of `x` is passed. A template for the problem definition file is:

```
function out1=probdef(flag,s,x,Vs,additional parameters)
switch flag
case 'x'
  out1 = optimal value of the control
case 'f'
  out1 = reward function
case 'g'
  out1 = drift term in state transition equation
case 'sigma'
  out1 = diffusion term in state transition equation
case 'rho'
  out1 = discount rate
end
```

A simplified (one-dimensional state and action) version of the solver is provided below. This version assumes that the optimal control can be explicitly calculated.

```
% SCSOLVE1 Solves stochastic control problems
% Simplified to handle the single state, single control case
function cv=scsolve1(model,fspace,snodes,v0);
  maxiters=100;
  tol=sqrt(eps);
% Compute part of collocation matrix that does not depend on x
  u=ones(1,fspace.n);
  rho=feval(scfile,'rho',s,[],[],varargin{:});
  Phi0=funbas(fspace,s,0);
  B=rho(:,u).*Phi0;
  sigma=feval(scfile,'sigma',s,[],[],varargin{:});
  sigma=0.5*sigma.*sigma;
  B=B-sigma(:,u).*funbas(fspace,s,2);
% The part of the collocation matrix that does depend on x
  Phi1=funbas(fspace,s,1);
```

```
% Initialize coefficient and control vectors
  if isempty(v0)
    c=zeros(fspace.n,1);
    v0=zeros(fspace.n,1);
  else
    c=Phi0\v0;
  end
  v=v0;
% Policy function iteration loop
  for i=1:maxiters
    Vs=Phi1*c;
    x=feval(scfile,'x',s,[],Vs,varargin{:});
    f=feval(scfile,'f',s,x,[],varargin{:});
    g=feval(scfile,'g',s,x,[],varargin{:});
    c=(B-g(:,u).*Phi1)\f;
    v0=v;
    v=Phi0*c;
    e=max(abs(v-v0));
    if e<tol, break; end
  end
  if i>=maxiters,                              % print warning message
    disp(['Algorithm did not converge. Maximum error: ' num2str(e)]);
  end
```

If no explicit solution for the optimal control exists, the problem definition file must return derivative information that can be used to numerically solve for the optimal control. In this case the a template for the problem definition file is:

```
function [out1,out2,out3]=probdef(flag,s,x,Vs,additional parameters)
switch flag
case 'f'
  out1 = reward function
  out2 = derivative of the reward function with respect to x
  out3 = second derivative of the reward function with respect to x
case 'g'
  out1 = drift term in state transition equation
  out2 = derivative of the drift function with respect to x
  out3 = second derivative of the drift function with respect to x
case 'sigma'
  out1 = diffusion term in state transition equation
case 'rho'
```

```
  out1 = discount rate
end
```

**Example: Renewable Resource Management**

The problem of optimally managing a renewable resource was discussed beginning on page 345. There are a total of 9 parameters in the model, $\alpha$, $\beta$, $K$, $b$, $\eta$, $c$, $\gamma$, $\rho$ and $\sigma$. The problem definition file for this example is provided below. It is complicated by the need to handle separately the cases in which $\eta = 1$ or $\beta = 0$ in order to avoid division by 0.

```
out1=psc01(flag,s,x,Vs,alpha,beta,K,b,eta,C,gamma,sigma,rho)
switch flag
case 'x'
  Cost=C*s.^(-gamma);
  out1=b*(Cost+Vs).^(-eta);
case 'f'
  Cost=C*s.^(-gamma);
  if eta~=1                          % handle demand elasticity <> 1
    factor1=1-1/eta;                 % case separately to avoid
    factor0=b.^(1/eta)/factor1;      % division by 0; see iteration loop
    out1=factor0*x.^factor1-Cost.*x;
  else                               % demand elasticity = 1
    out1=b*log(x)-Cost.*x;
  end
case 'g'
  if beta~=0                                 % need to handle beta=0
    Growth=alpha/beta*s.*(1-(s/K).^beta);    % case separately
  else                                       % to avoid division by 0
    Growth=alpha*s.*log(K./s);
  end
  out1=Growth-x;
case 'sigma'
  out1=sigma*s;
case 'rho'
  out1=rho+zeros(size(s,1),1);
end
```

Notice that an explicit expression for the optimal control is used so there is no need to provide derivative information.

A script file using `scsolve` to obtain a solution is provided below. The parameter values used are $\alpha = 0.5$, $\beta = 1$, $K = 1$, $b = 1$, $\eta = 0.5$, $c = 0.1$, $\gamma = 2$, $\rho = 0.05$, and $\sigma = 0.1$. The natural state space is $S \in [0, \infty)$. As discussed below, an approximation is feasible only over a bounded interval. The demonstration uses a piecewise linear approximation with 75 breakpoints, evenly spaced on the interval $[0.2, 1.2]$.

```
% Set parameter values
  beta=1;     eta=0.5;     gam=2;
  alpha=0.5;    K=1;         b=1;
  C=5;        rho=0.05;  sigma=0.1;
  beta=1;     eta=0.5;     gam=2;
% Define model variable
  model.func='psc01';
  model.params={alpha,beta,K,b,eta,C,gam,sigma,rho};
% Define the approximating family and nodes
  lo=0.2;  hi=1.2; n=75;
  fspace=fundefn('plin',n,lo,hi);
  s=funnode(fspace);
% Call the stochastic control solver
  cv=scsolve(model,fspace,s);
```

An explicit solution exists for the parameter values used in the demonstration, as displayed in Table 10.1 on page 347. The demonstration file plots the relative error functions over the range of approximation for the marginal value function and optimal control (i.e., $1 - \hat{V}/V$ and $1 - \hat{x}/x$). The resulting plot is displayed in Figure 11.4. It can be seen that the errors in both functions are quite small, except at the upper end of the range of approximation.

As with all problems involving an unbounded state space, a range of approximation must be selected. For infinite horizon problems, the general rule of thumb is that the range should include events that the stationary (long-run) distribution places a non-negligible probability of occurrence. In this case, there is little probability that the resource stock, optimally harvested, will ever be close to the biological carrying capacity of $K$. It is also never optimal to let the stock get too small.[4]

The renewable resource problem has a natural lower bound on $S$ of 0. It turns out, however, that 0 is a poor choice for the lower bound of the approximation because the value function goes to $-\infty$ and the marginal value function to $\infty$ as $S \to 0$. Such behavior is extremely hard to approximate with splines or polynomials. Inclusion

[4]For the parameters values used in the example, the long run distribution has a known closed form. It can be shown that the probability of values of $S$ less that 0.2 or greater than 0.8 are effectively 0.

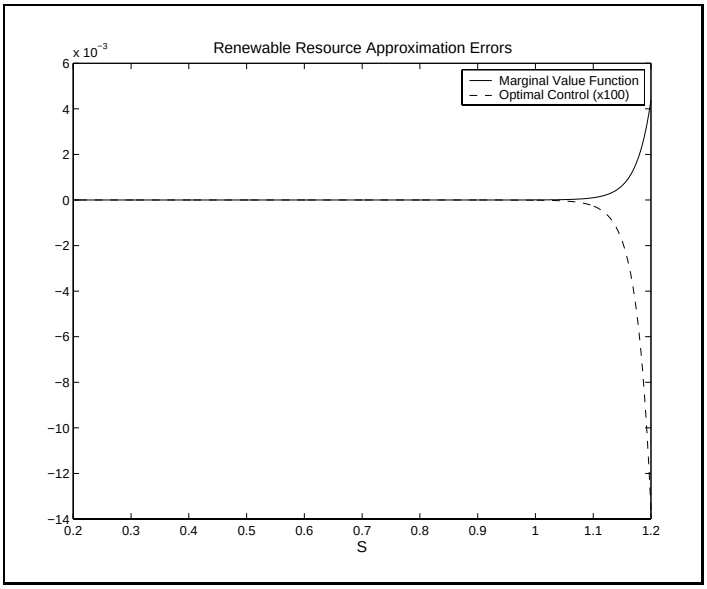


Figure 11.4

of basis functions that exhibit such behavior is possible but requires more work. It is also unnecessary, because the approximation works well with a smaller range of approximation.

In practice, of course, we may not know the stationary distribution or how large the errors are. There are several methods to address whether the approximation choices are good ones. First, check whether the approximation provides reasonable results. Second, check the residual function at inter-nodal values of $S$. Third, check whether increasing the range of approximation changes the value of the approximating function in a significant manner.

Before leaving this example, a technique for improving the accuracy of a solution will be demonstrated. For a number of reasons, a piecewise linear approximation with finite difference derivatives is a fairly robust choice of approximating functions. It is a shape preserving choice, in the sense that an approximation to a monotonic and/or convex function will also be monotonic and/or convex. This makes it far more likely that the iterations will converge from arbitrary starting values.

The piecewise linear approximation, however, requires a large number of collocation nodes to obtain accurate approximations. It may therefore be useful to obtain a rough solution with a piecewise linear approximation and use that approximation as a starting point for a second, more accurate approximation. The following code fragment could be appended to the script file on page 414 to obtain a polynomial approximation on the same interval.

```
s2=linspace(lo,hi,101)';
fspace2=fundefn('cheb',35,lo,hi);
cv2=scsolve(model,fspace,s2,funeval(cv,fspace,s2));
```

This code fragment defines a new family of approximating functions and uses the previously computed solution to provide starting values. The polynomial approximation with 35 collocation nodes produces an approximation with a relative error of under $10^{-10}$ except at the upper end of the range of approximation (above 0.9).

**Example: Optimal Growth**

The neoclassical optimal growth model, discussed beginning on page 353, solves

$$\max_{C(t)} \int_0^\infty e^{-\rho t} U(C) dt,$$

subject to the state transition function $K' = q(K) - C$. This problem could be solved using the `scsolve` function; we leave this as an exercise (page 452). The optimal control ($C$) satisfies the Euler condition

$$-\frac{U'(C)}{U''(C)}(q'(K) - \rho) = (q(K) - C)C'(K),$$

a first order differential equation. The solution $C(K)$ passes through the steady state point $(K^*, C^*)$ which simultaneously solves $dK/dt = 0$ and the Euler condition:

$$q'(K^*) = \rho$$

$$C^* = q(K^*).$$

Our solution approach requires that three functions be defined. The first is $q(K)$, the second is $s(K) = q'(K) - \rho$, which measures the excess marginal productivity over the discount rate and the third is the absolute risk aversion function, $r(C) = -U''(C)/U'(C)$. The steady state capital stock solves $s(K) = 0$.

The optimal control can be approximated on the interval $[a, b]$ with a function $\phi(K)c$. The Euler condition is used to form the residual function

$$0 \approx e(K) = \Big(q(K) - \phi(K)c\Big)\phi'(K)c - \frac{s(K)}{r(\phi(K)c)}.$$

This equation can be solved at a set of nodal values, $k_i$, simultaneously with the boundary condition that $\phi(K^*)c - C^* = 0$.

To make the problem concrete, define

$$q(K) = \alpha \ln(K+1) - \delta K,$$

implying that

$$s(K) = q'(K) - \rho = \frac{\alpha}{K+1} - \delta - \rho.$$

The steady state capital stock is therefore $K^* = \dfrac{\alpha}{\delta+\rho} - 1$. As the units in which $K$ is denominated are arbitrary, it is convenient to set $K^* = 1$ by defining $\alpha = 2(\delta+\rho)$. Furthermore, let the utility function be of the constant relative risk aversion (CRRA) form, $U(C) = (C^{1-\gamma} - 1)/(1-\gamma)$, which implies that $r(C) = \gamma/C$. The relative risk aversion parameter, $\gamma$, takes on values between 0 and 1, with 0 implying risk neutrality and 1 resulting in logarithmic utility. Specific parameter values are $\rho = 0.05$, $\delta = 0.02$, $\alpha = 2(\rho+\delta) = 0.14$ and $\gamma = 0.5$.

A script file to solve the problem is displayed in Code Box 1. The code uses "inline" functions to define $q(K)$, $s(K)$ and $r(C)$ at the beginning of the file. These three functions can be thought of as the parameters defining the model and can be changed without altering the rest of the code to explore the implications of alternative parameters.

The next step is to determine the steady state values. We pass $s(K)$ to rootfinding algorithm `broyden` to obtain $K^*$. This is a bit gratuitous in this example given that we already know that $K^* = 1$, but it illustrates how $K^*$ can be found if $s(K) = 0$ cannot be solved directly. Given the model parameters, the steady state consumption rate is $C^* = 0.077$ or 7.7% of the capital stock.

The next step is to set up the collocation problem. We use a Chebyshev polynomial approximation of degree $n = 20$. To impose the boundary condition, it is necessary, however, to solve the Euler equation at $n-1$ nodes and impose the additional restriction that $\phi(K^*)c = C^*$. In practice, it turns out that it is useful to impose the additional restriction that no consumption can occur when the capital stock is 0: $C(0) = 0$. Thus, we use $n-2$ collocation nodes on the interval $[0, 2K^*]$ for the Euler equation, plus the two boundary conditions. The Euler equation nodes used are simply the Chebyshev nodes for degree $n-2$. These $n-2$ nodes are then appended to 0 and $K^*$ and the basis matrices for the function and its derivative are computed ($\Phi_0$ and $\Phi_1$). An initial coefficient vector is fit to the straight line through $(0,0)$ and $(K^*, C^*)$ using the `funfitxy` procedure.

To solve the collocation problem, we will need to pass a residual function to a rootfinding algorithm. The residual function is defined in a separate file shown in Code Box 2. The residual file takes values of $c$, the approximation coefficient vector, and returns the collocation residual vector. A number of precomputed values are

**Code Box 11.1: Neoclassical Growth Model**

```
% DEMSC03 Neo-classical Optimal Growth Problem (cont. time)

% Define Problem Parameters
  % q(K)=alpha*log(K+1) - delta*K
    q=inline('0.14*log(K+1) - 0.02*K','K');
  % s(K)=q'(K)-rho
    s=inline('0.14./(K+1) - 0.02 - 0.05','K');
  % r(C)=-U''(C)/U'(C)
    r=inline('0.5./C','C');
% Find steady state solution
  Kstar=broyden(s,1);
  Cstar=q(Kstar);
  disp('Steady State Capital and Consumption')
  disp([Kstar Cstar])
% Define nodes and basis matrices
  n=20;
  a=0; b=2*Kstar;
  cdef=fundef({'cheb',n-2,a,b});
  K=[0;Kstar;funnode(cdef)];
  cdef=fundef({'cheb',n,a,b});

  Phi0=funbas(cdef,K,0);
  Phi1=funbas(cdef,K,1);
  svals=s(K);
  qvals=q(K);
% Get initial value of C, linear in K
  k=linspace(cdef.a,cdef.b,301)';
  c=funfitxy(cdef,k,Cstar/Kstar*k);
  c=broyden('fsc03',c,[],r,svals,qvals,Phi0,Phi1,Cstar);
% Plot optimal control function
  figure(1)
  C=funeval(c,cdef,k); C(1)=0;
  plot(k,C,Kstar,Cstar,'*')
  title('Optimal Consumption Rule')
  xlabel('K')
  ylabel('C')
% Plot residual function
  figure(2)
  dC=funeval(c,cdef,k,1);
  warning off                    % avoid divide by zero warning
  e=(q(k)-C).*dC-s(k)./r(C);
  warning on
  plot(k,e)
  title('Growth Model Residual Function')
  xlabel('K')

prtfigs(mfilename)
```

passed to this function as auxiliary parameters. These include the basis matrices, $\Phi_0$ and $\Phi_1$, the steady state control, $C^*$, and the risk aversion function $r(C)$. The values of $q(K)$ and $s(K)$ are also needed to compute the residual function. Unlike the values of $r(C)$, however, these do not change with $c$, so they can be precomputed and passed as vectors rather than "inline" functions. The residual function first computes the residuals for the Euler equation and then alters the first two values to impose the boundary conditions at $K = 0$ and $K = K^*$.

**Code Box 11.2: Residual Function for Neoclassical Growth Model**

```
% FSC03 Residual function for neoclassical optimal growth problem
% See DEMSC03 for usage
function e=fsc03(c,r,svals,qvals,Phi0,Phi1,Cstar)
C=Phi0*c;
dC=Phi1*c;
warning off
e=(qvals-C).*dC-svals./feval(r,C);
warning on
e(1)=C(1);
e(2)=C(2)-Cstar;
```

After using `broyden` to find the coefficient vector for the optimal consumption function, the demonstration file then produces a plot of this function, which is shown in Figure 11.5. The steady state equilibrium point $(K^*, C^*)$ is marked with a "*". The demonstration code also plots the residual function defined by the Euler equation. This provides an estimate of the error of the approximation. As seen in Figure 11.6 the residual function is relatively small, even with a degree 20 approximation.

## 11.3 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. In their most simple form, with a single state variable in an infinite horizon situation, these problems involve solving a second order linear differential equation of the form

$$\rho(S)V(S) = f(S) + \mu(S)V'(S) + \tfrac{1}{2}\sigma^2(S)V''(S), \tag{2}$$

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. As discussed in Section 6.8.3

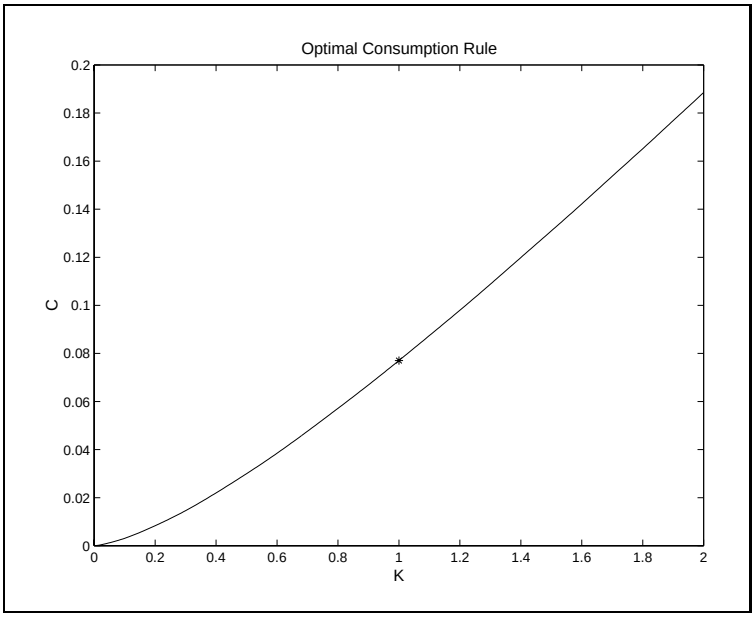


Figure 11.5

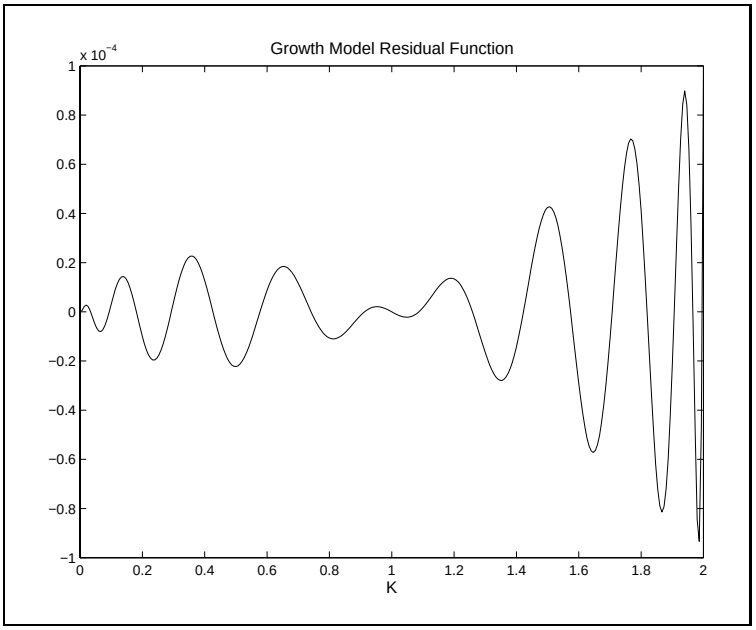


Figure 11.6

(page 164), one can approximate the solution using a function parameterized by an $n$-vector $c$, $V(S) \approx \phi(S)c$, with $c$ 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 location of the upper boundary, $b$, is unknown but is known to satisfy $V'(b) = h_b$, 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.

A general strategy treats the solution interval $[a, b]$ as known, finds an approximate solution on this interval using the differential equations along with $V(a) = g_a$ and $V(b) = g_b$. This allows the residual functions $V'(a) - h_a$ and $V'(b) - h_b$ to be defined. These are passed to a rootfinding algorithm to determine the location of the free boundaries.

A problem with this method, however, is that the approximating functions we use typically define the function in terms of its boundaries. Here, however, the interval on which the approximating function is to be defined is unknown. Fortunately, this problem is easily addressed using a change in variable. To illustrate, consider first the case in which $b$ is unknown and, for simplicity, $a = 0$. Define

$$y = S/b,$$

so the differential equation is defined on $y \in [0, 1]$. Define the function $v(y)$ such that

$$V(S) = v(y).$$

Using the chain rule it can be seen that

$$V'(S) = v'(y)\frac{dy}{dS} = \frac{v'(y)}{b}$$

and

$$V''(S) = v''(y)\left(\frac{dy}{dS}\right)^2 + v'(y)\frac{d^2y}{dS^2} = \frac{v''(y) - v'(y)}{b^2}.$$

Inserting these definitions into (2) demonstrates that the original problem is equivalent to

$$\rho(by)v(y) = f(by) + \frac{\mu(by)}{b}v'(y) + \frac{\sigma^2(by)}{2b}v''(y), \tag{3}$$

for $y \in [0,1]$, with

$$\begin{aligned} v(0) &= g_a, \\ v(1) &= g_b, \end{aligned} \tag{4}$$

and

$$v'(1) = bh_b. \tag{5}$$

An approximation, $v(y;b) = \phi(y)c(b)$ can be found for arbitrary values of $b$ using (3) and (4). The optimal choice of $b$ can then be determined using a root finding algorithm to solve (5) using $\phi'(1)c(b) - bh_b = 0$. More complicated problems will use a similar strategy, but may need to handle multiple value functions, multiple boundaries and/or boundaries that are curves or surfaces in the state space rather than isolated points. After illustrating the basic approach in the simplest of cases, we discuss ways to handle the complications.

Before proceeding, we should point out that there is a vast and growing literature on free boundary problems. Our goal here is rather modest. We want to present a framework that can solve such problems, while building on the methods already developed in this book. As such, we are not attempting to provide the most efficient approaches to specific problems, but rather ones that will provide useful answers without a lot of special coding. We also make no attempt to provide general solvers for these models but rather demonstrate by example how they can be solved.

**Example: Asset Replacement**

The infinite time horizon resetting problem with a single state variable is perhaps the simplest example of a free boundary problem to solve. Recall the asset replacement problem (page 359) in which an asset's productivity depends on its age, $A$, yielding a net return of $Q(A)P$. The Bellman equation is

$$\rho V(A) = Q(A)P + V'(A),$$

and applies on the range $A \in [0, A^*]$, where $A^*$ is the optimal replacement age. The asset can be replaced by paying a fixed cost C. The boundary conditions are given by the value matching condition:

$$V(0) = V(A^*) + C$$

and the optimality (smooth pasting) condition:

$$V'(A^*) = 0.$$

We first transform the problem in terms of $y = A/A^*$, defining $v(y) = V(A)$ and, therefore, $v'(y) = V'(A)A^*$. Thus, the Bellman's equation can be expressed in terms of $y$:

$$\rho v(y) = Q(A^*y)P + v'(y)/A^*$$

with boundary conditions $v(0) - v(1) = C$ and $v'(1) = 0$.

We approximate the value function using $v(y) \approx \phi(y)c$, for $y \in [0, 1]$, where $c$ is an $n$-vector of coefficients. Then the Bellman's equation can be expressed as the residual function

$$[\rho\phi(y) - \phi'(y)/A^*]c = Q(A^*y)P.$$

The Bellman equation is solved with equality at $n-1$ nodal points on $[0, 1]$ simultaneously with the value-matching condition $[\phi(0) - \phi(1)]c = C$. This is a linear system in $c$ and hence can be solved directly.

The system of $n-1$ collocation equations plus the value-matching condition provides a value of $c$ that is conditional on the choice of the free boundary $A^*$. This choice is optimal when $\phi'(1)c(A^*) = 0$; this is a single equation in a single unknown and is easily solved using a nonlinear rootfinding algorithm. The method is demonstrated below for the case in which $Q(A) = 1 + 0.05A - 0.003A^2$, $P = 2$, $C = 3$ and $\rho = 0.1$.

```
% DEMFB01 Asset Replacement Demonstration
% Define parameters
  Q=inline('1+.05*A-.003*A.^2','A');
  P=2;
  C=3;
  rho=0.1;

% Define nodes and basis matrices
  n=15;
  cdef=fundefn('cheb',n-1,0,1);
  y=funnode(cdef);
  cdef=fundefn('cheb',n,0,1);
  rPhi0=funbas(cdef,y);                 % rho*phi(y)
  Phi1=funbas(cdef,y,1);                % phi'(y)
  phiVM=funbas(cdef,0)-funbas(cdef,1);  % phi(0)-phi(1) for value matching
  phiSP=funbas(cdef,1,1);               % phi'(1)       for smooth pasting

% Call rootfinder
  Astar=100;      % initial guess
```

```
Astar=broyden('ffb01',Astar,[],Q,P,C,y,rPhi0,Phi1,phiVM,phiSP);
[e,c]=ffb01(Astar,Q,P,C,y,rPhi0,Phi1,phiVM,phiSP);
```

The nodal values of $y$ are defined as the standard Chebyshev nodes for degree $n-1$ rather then for degree $n$, to accommodate the addition of the value-matching condition. Furthermore, in addition to the basis matrices for the Bellman's equation ($\Phi_0$ and $\Phi_1$), we precompute basis vectors ($1 \times n$) for the value-matching and smooth-pasting conditions. The latter vectors are defined as $\phi_{VM} = \phi(0) - \phi(1)$ and $\phi_{SP} = \phi'(1)$, respectively.

After setting $A^*$ to an initial guess of 100, `broyden` is called to solve for the optimal $A^*$. This requires that a separate residual function file be defined:

```
function [e,c]=ReplaceRes(Astar,Q,P,C,y,rPhi0,Phi1,phiVM,phiSP)
B=[rPhi0-Phi1./Astar;phiVM];
b=[feval(Q,Astar*y)*P;C];
c=B\b;
e=phiSP*c;
```

The residual function first determines $c(A^*)$ by solving

$$\begin{bmatrix} \rho\phi(y_1) - \phi'(y_1)/A^* \\ \cdots \\ \rho\phi(y_{n-1}) - \phi'(y_{n-1})/A^* \\ \phi(0) - \phi(1) \end{bmatrix} c = \begin{bmatrix} Q(A^* y_1)P \\ \cdots \\ Q(A^* y_{n-1})P \\ C \end{bmatrix}.$$

It then returns the value $e = \phi'(1)c(A^*)$. It will also return the computed value of $c$, which can then be used to evaluate $v(y)$ and hence $V(A)$.

A plot of $V(A)$ is shown in Figure 11.7. The value-matching condition appears in the figure as the difference of $C = 3$ between $V(0)$ and $V(A^*)$ ($A^* \approx 17.64$). The smooth-pasting condition appears as the 0 slope of $V$ at $A^*$.

**Example: Investment Timing**

In the previous example, a state variable was controlled in such a way as to keep it within a region of the state space bounded by a free boundary. In other problems, the state can wander outside of the region defined by the free boundary but the problem is either known or has no meaning outside of the region. In such case the value function need only be approximated on the region of interest, using appropriate boundary conditions to define both the value function and the boundary itself. From a computational perspective such problems require no new considerations.

To illustrate, 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

$$dS = \alpha(m - S)Sdt + \sigma S dz.$$

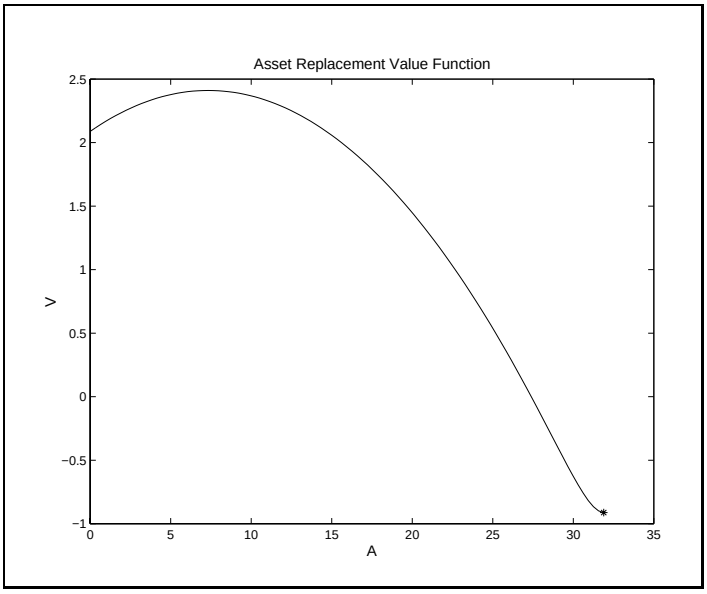


Figure 11.7

This process can be shown to have a mean reverting rate of return, with long-run mean $m$ (see Appendix A, Section A.5.2). When the investment is made, it has net 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

$$\tfrac{1}{2}\sigma^2 z^2 v''(z) + \alpha(m - zS^*)zv'(z) - rv(z) = 0, \tag{6}$$

on the interval $[0, 1]$, with $v(0) = 0$, $v(1) = S^* - I$, and $v'(1) = S^*$. To solve the problem we use an approximation of the form $v(z) \approx \phi(z)c$. 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 (6)

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

$$c(S^*) = B^{-1} \left[ \begin{array}{c} 0_{n-1} \\ S^* - I \end{array} \right].$$

Given $c$ we can define a residual function in one dimension to solve for $S^*$ using the smooth-pasting condition:

$$e(S^*) = S^* - \phi'(1)c(S^*).$$

This approach works well in some cases but this example has one additional problem that must be addressed. 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 (6) 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. MATLAB code solving the problem using function approximation is displayed in Code Boxes 3 and 4.

This particular problem has a partially known solution. It can be shown that the solution can be written as

$$V(S) = AS^{\beta} H(\phi S; \beta, \kappa),$$

### Code Box 11.3: Solution Function for Optimal Investment Problem

```
% OPTINVEST Solves the optimal investment problem with mean-reverting return
% Finds the optimal investment rule for a project that has return process
%   dS=alpha(m-S)dt + sigma*SdW
% USAGE
%   [e,vstar,V,dV]=OptInvest(v,r,alpha,m,sigma,I,n);
% INPUTS
%   r : interest rate
%   alpha, m, sigma : return process paramters
%   I : fixed investment cost
%   n : number of nodes used for Chebyshev collocation
% OUTPUTS
%   Sstar  : the trigger return level (invest if S>=Sstar)
%   c      : coefficients for value function approximation
%   fspace : function family definition structure

function [Sstar,c,fspace]=optinvest(r,alpha,m,sigma,I,n)
  if nargin<6 | isempty(n),  n=50; end
  fspace=fundefn('cheb',n-1,0,1);
  t=funnode(fspace);                        % nodal points on [0,1]
  fspace=fundefn('cheb',n,0,1);
  Phi0=funbas(fspace,t);
  Phi1=funbas(fspace,t,1);
  u=ones(1,n);
  tt=t.*t;
  B=(sigma.^2/2)*tt(:,u).*funbas(fspace,t,2) + alpha*m*t(:,u).*Phi1 - r*Phi0;
  B=[B;funbas(fspace,[1])];
  B1=[alpha*tt(:,u).*Phi1;zeros(1,n)];   % basis matrix for VSTAR part
  phi11=funbas(fspace,1,1);

  Sstar=2*m;
  Sstar=broyden('optinvestr',Sstar,[],I,B,B1,phi11,n);
  [e,c]=optinvestr(Sstar,I,B,B1,phi11,n);
```

### Code Box 11.4: Residual Function for Optimal Investment Problem

```
% OPTINVESTR Residual function for optimal investment problem
function [e,c]=optinvestr(Sstar,I,B,B1,phi11,n)

  c=(B-Sstar*B1)\[zeros(n-1,1);Sstar-I];
  e=Sstar-phi11*c;
```

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

$$\beta=\frac{1}{2}-\frac{\alpha m}{\sigma^2}+\sqrt{\left(\tfrac{1}{2}-\frac{\alpha m}{\sigma^2}\right)^2+\frac{2r}{\sigma^2}}$$

$$\kappa=1+2\sqrt{\left(\tfrac{1}{2}-\frac{\alpha m}{\sigma^2}\right)^2+\frac{2r}{\sigma^2}}$$

$$\phi=\frac{2\alpha}{\sigma^2}\,.$$

The problematic parrameters 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:

$$AS^{*\beta}H(\phi S^*;\beta,\kappa)-(S^*-I)=0$$

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[5]

$$S^*-\beta(S^*-I)\left(1+\frac{\phi}{\kappa}\frac{H(\phi S^*;\beta+1,\kappa+1)}{H(\phi S^*;\beta,\kappa)}S^*\right)=0,$$

a simple root finding problem in a single variable (see the toolbox file `optinvest2.m`).

A demonstration file `demfb02.m` uses both approaches with the parameter values $r=0.04$, $\alpha=0.5$, $m=1$, $\sigma=0.2$ and $I=1$. Wit the first method, the problem is solved using a Chebyshev approximation of degree $n=50$, with the lower end point condition not imposed. The code produces Figures 11.8 and 11.9, showing the value function and the approximation errors. The parameters values imply $\beta=0.0830$, making the value function near the lower bound rise very steeply. The approximation displays some slight wiggling in this region, illustrating the difficulties in fitting the value function at the lower end. The computation of the location of the free boundary is not very sensitive to these problems, however, and is computed with an error of less than $10^{-4}$.

[5]Notice from the series expansion that the derivative of $H$ is given by

$$H'(x;\beta,\kappa)=\frac{\beta}{\kappa}\sum_{i=0}^{\infty}\frac{\Gamma(\beta+i+1)\Gamma(\kappa)x^i}{\Gamma(\beta)\Gamma(\kappa+i+1)i!}=\frac{\beta}{\kappa}H(x;\beta+1,\kappa+1).$$

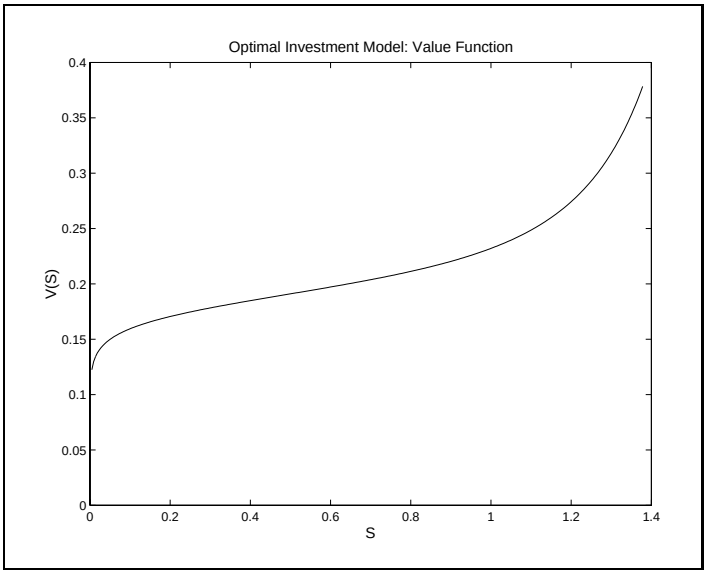


Figure 11.8

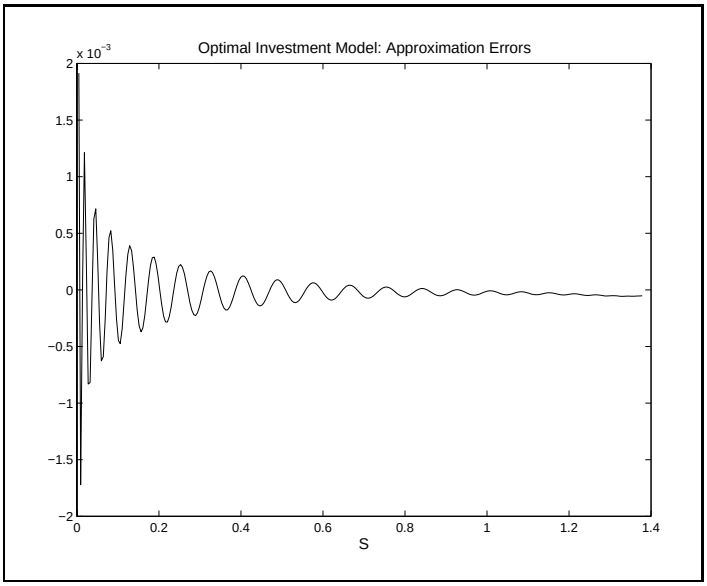


Figure 11.9

### 11.3.1 Multiple Value Functions

Problems with binary states that can be exited and reentered, as is the case with entry/exit and stochastic bang-bang problems, 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:

$$x_l \leq x \leq x_u.$$

In 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 where $f_1 = g_1 V_S$ (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 $S^*$ (usually 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 $V(S)$. To solve this we will use 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 $\phi_i(S)c_i$, for $i = l, u$ and define the function $B_i(S)$ as

$$B_i(S) = \rho\phi_i(S) - [g_0(S) + g_1(S)x_i]\,\phi_i'(S) - \tfrac{1}{2}\sigma^2(S)\phi_i''(S)$$

The $c_i$ can be determined by making

$$B_i(S)c_i - [f_0(S) + f_1(S)x_i] = 0$$

at a selected set of collocation nodes, together with the boundary conditions and

$$\begin{array}{ll} \phi_l(S^*)c_l - \phi_u(S^*)c_u = 0 & \text{(value matching)} \\ \phi_l'(S^*)c_l - \phi_u'(S^*)c_u = 0 & \text{(smooth pasting).} \end{array}$$

Determining the $c_i$ for some guess of $S^*$, therefore, amounts to solving a system of linear equations (assuming any additional boundary conditions are linear in $V$). Once the $c_i$ are determined, the residual

$$r(S^*) = f_1(S^*) + g_1(S^*)\phi_l'(S^*)c_l$$

can be computed. The optimal value of $S^*$ is then chosen to make $r(S^*) = 0$.

**Example: Optimal Fish Harvest**
In the optimal fish harvesting problem (page 345) the value function solves the coupled PDE

$$\rho V = \begin{cases} \alpha S(1 - S/K)V_S + \frac{1}{2}\sigma^2 S^2 V_{SS} & \text{for } S < S^* \\ PHS + (\alpha S(1 - S/K) - HS)\,V_S + \frac{1}{2}\sigma^2 S^2 V_{SS} & \text{for } S > S^* \end{cases}$$

with $S^*$ determined by $P = V_S(S^*)$ and continuity of $V$ and $V_S$ at $S^*$. For simplicity, we impose the scale normalization $P = K = 1$ (by choosing units for money and fish quantity).

To solve this problem we first transform the state variable by setting

$$y = \ln(S) - \ln(S^*).$$

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 $y = 0$.

The transformation necessitates rewriting the value function in terms of $y$, say as $v(y)$. The transformation implies that

$$S = S^* e^y,$$

$$SV_S(S) = v_y(y)$$

and

$$S^2 V_{SS}(S) = v_{yy}(y) - v_y(y).$$

The transformed Bellman equation with the scale normalizations is

$$\rho v = \begin{cases} \frac{1}{2}\sigma^2 v_{yy} + \left(\alpha(1 - S^* e^y) - \frac{1}{2}\sigma^2\right) v_y & \text{for } y < 0 \\ \frac{1}{2}\sigma^2 v_{yy} + \left(\alpha(1 - S^* e^y) - \frac{1}{2}\sigma^2 - H\right) v_y + S^* H e^y & \text{for } y > 0 \end{cases}.$$

It will be useful to rewrite this to isolate the $S^*$ terms

$$\begin{bmatrix} \rho v - \left(\alpha - \frac{1}{2}\sigma^2\right) v_y - \frac{1}{2}\sigma^2 v_{yy} \end{bmatrix} + S^* \alpha e^y v_y = 0 \quad \text{for } y < 0$$
$$\begin{bmatrix} \rho v - \left(\alpha - \frac{1}{2}\sigma^2 - H\right) v_y - \frac{1}{2}\sigma^2 v_{yy} \end{bmatrix} + S^* \alpha e^y v_y = S^* H e^y \quad \text{for } y > 0.$$

The two functions are coupled by imposing continuity of $v$ and $v_y$ at $y = 0$. Technically there are also boundary conditions as $y$ goes to $-\infty$ and $\infty$ , but we will ignore these for the time being.

Now let's approximate the two functions using $\phi_l(y)c_l$ and $\phi_u(y)c_u$, where the $\phi_i$ are $n_i$-element basis vectors and the $c_i$ are the coefficients associated with these bases. For a specific guess of $S^*$, the Bellman equation can be written

$$\begin{array}{ll} D_l(y)c_l + S^* \left[\alpha e^y \phi_l'(y)\right] c_l = 0 & \text{for } y < 0 \\ \left[D_u(y) + H\phi_u'(y)\right] c_u + S^* \left[\alpha e^y \phi_u'(y)\right] c_u = S^* H e^y & \text{for } y > 0 \end{array}.$$

where $D_i(y) = \rho\phi_i(y) - \left(\alpha - \frac{1}{2}\sigma^2\right)\phi_i'(y) - \frac{1}{2}\sigma^2\phi_i''(y)$. Evaluating this expression at a set of nodes, $y_l \in [a, 0]$, and $y_u \in [0, b]$, where $a$ and $b$ are arbitrary upper and lower bounds, with $a < 0$ and $b > 0$.

The boundary conditions at $y = 0$ 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 $y_l$ and $y_u$ to have $n_l - 1$ and $n_u - 1$ elements, respectively, this yields the $n_l + n_u$ system of linear equations:

$$\left( \begin{bmatrix} B_l & 0 \\ 0 & B_u \\ \phi_l(0) & -\phi_u(0) \\ \phi_l'(0) & -\phi_u'(0) \end{bmatrix} + S^* \begin{bmatrix} D_l & 0 \\ 0 & D_u \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \right) \begin{bmatrix} c_l \\ c_u \end{bmatrix} = \begin{bmatrix} 0 \\ S^* H e^{y_u} \\ 0 \\ 0 \end{bmatrix}.$$

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 $c$ and therefore 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 $S^*$:

$$c(S^*) = (B + S^* D)^{-1} S^* f. \tag{7}$$

The optimal $S^*$ is then determined by solving the (non-linear) equation

$$S^* - \phi_l'(0) c_l(S^*) = 0. \tag{8}$$

A MATLAB implementation is displayed in Code Box 5. The procedure `fishh` defines the approximating functions and precomputes the matrices needed to evaluate (7) and (8). The actual evaluation of these equations is performed by the auxiliary procedure `fishhr` displayed in Code Box 6, which is passed to the rootfinding algorithm `broyden` by `fishh`. A script file which computes and plots results is given in Code Box 7; this was used to produce Figures 11.10-11.13.

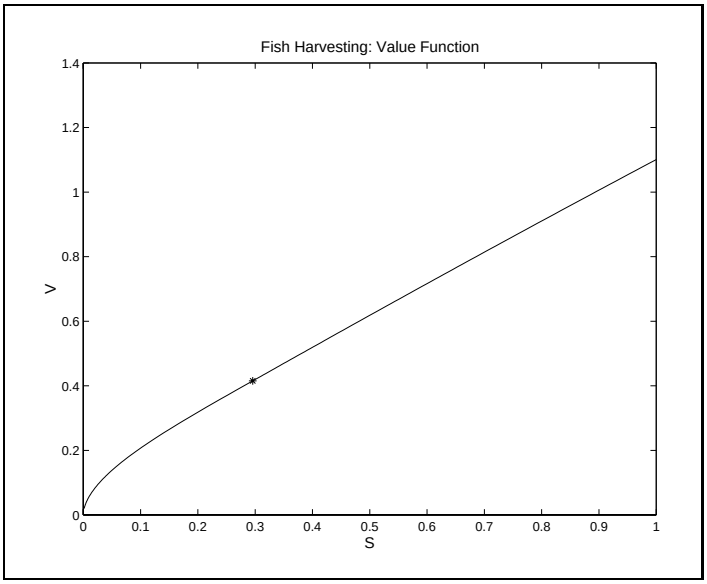


Figure 11.10

The procedure `fishh` allows additional constraints to be imposed at the lower and upper bounds of the approximating interval ($a$ and $b$). By setting `nbl` to 2, the additional constraint that $v_l''(a) = 0$ is imposed. Similarly, by setting `nbu` to 2, the additional constraint that $v_u''(b) = 0$ is imposed. Although neither is necessary to

**Code Box 11.5: Collocation File for Fish Harvesting Problem**

```
% FISHH Solution function for fish harvesting problem
% See DEMFB03 for demonstration
function [sstar,cl,cdefl,cu,cdefu]=fishh(rho,alpha,sigma,H)

% Set up basis matrices
  a=log(0.005);        % lower bound
  b=log(10);           % upper bound
  nl=25;   nu=15;      % number of nodes for functions
  nbl=2;   nbu=1;      % number of boundary constraints on functions
  cdefl=fundef({'cheb',nl-nbl,a,0});
  cdefu=fundef({'cheb',nu-nbu,0,b});
  yl=funnode(cdefl);
  yu=funnode(cdefu);
  cdefl=fundef({'cheb',nl,a,0});
  cdefu=fundef({'cheb',nu,0,b});
  eyl=exp(yl);
  eyu=exp(yu);
  Dl=funbas(cdefl,yl,1);
  Du=funbas(cdefu,yu,1);
  B=rho*funbas(cdefl,yl)...
       -(alpha-0.5*sigma.^2)*Dl...
       -(0.5*sigma.^2)*funbas(cdefl,yl,2);
  temp=rho*funbas(cdefu,yu)...
       -(alpha-0.5*sigma.^2-H)*Du...
       -(0.5*sigma.^2)*funbas(cdefu,yu,2);
  B=[B zeros(nl-nbl,nu);zeros(nu-nbu,nl) temp];
% Add boundary constraints
  B=[B; ...
     funbas(cdefl,0)   -funbas(cdefu,0); ...          % V continuous at y=0
     funbas(cdefl,0,1) -funbas(cdefu,0,1)];           % Vx continuous at y=0
  if nbl==2; B=[B;funbas(cdefl,a,2) zeros(1,nu)]; end % lower boundary
  if nbu==2; B=[B;zeros(1,nl) funbas(cdefu,b,2)]; end % upper boundary
% Basis for Vy
  D=[alpha*eyl*ones(1,nl).*Dl    zeros(nl-nbl,nu); ...
     zeros(nu-nbu,nl)            alpha*eyu*ones(1,nu).*Du; ...
     zeros(nbl+nbu,nl+nu)];
% RHS of DE residual function
  f=[zeros(nl-nbl,1);H*eyu;zeros(nbl+nbu,1)];
% Basis for residual function (Vy(0)=S*)
  phil10=funbas(cdefl,0,1);
% find the cutoff stock level
  sstar=broyden('fishhr',0.5*(1-rho/alpha),[],B,D,f,phil10,nl);
% Break apart the coefficient vector and create structures to return
  c=(B+sstar*D)\(sstar*f);
  cl=c(1:nl);
  cu=c(nl+1:end);
```

**Code Box 11.6: Residual File for Fish Harvesting Problem**

```
% FISHHR residual function for fish harvesting problem
% Used by FISHH

function [e,c]=fishhr(sstar,B,D,f,phil10,nl)
  c=(B+sstar*D)\(sstar*f);
  e=sstar-phil10*c(1:nl);
```

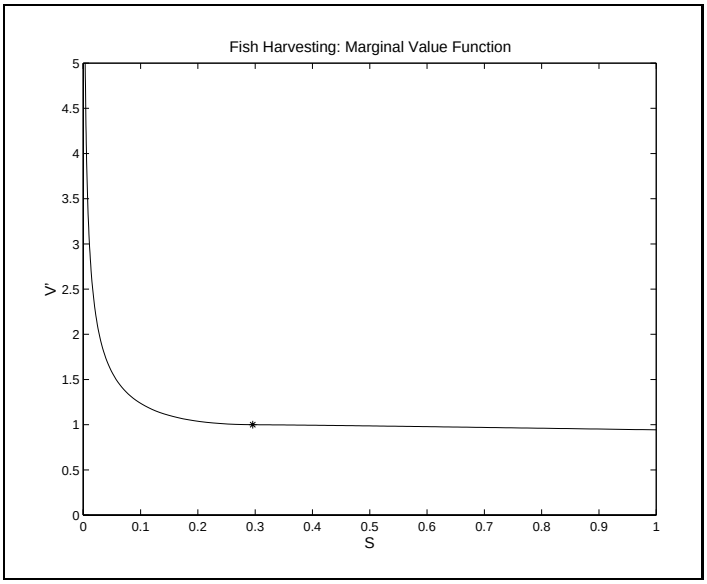


Figure 11.11

obtain useful results, the former is effective in enforcing sensible behavior in the value function and its derivatives for low stock levels.

Figure 11.10 illustrates a numerical approximation to the value function for the problem with $\rho = 0.05$, $\alpha = 0.1$, $\sigma = 0.2$ and $H = 1$. Figures 11.11 and 11.12 display the first and second derivatives of the value function. $S^*$ 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. Evidence of the quality of the approximation is provided by the plot of the residual function, shown

**Code Box 11.7: Script File for Fish Harvesting Problem**

```
% DEMFB03 Demo for fish harvesting problem
  rho=0.05;
alpha=0.1;
sigma=0.2;
    H=1;
[sstar,cl,cdefl,cu,cdefu]=fishh(rho,alpha,sigma,H);

%  CODE TO GENERATE PLOTS
a=cdefl.a; b=cdefu.b;
N=101;
yl=linspace(a,0,N)';
yu=linspace(0,log(1/sstar),N)';
s=sstar*exp([yl;yu]);

figure(1)
v=[funeval(cl,cdefl,yl);funeval(cu,cdefu,yu)];
plot(s,v,sstar,v(N),'*');
title('Fish Harvesting: Value Function');
xlabel('S');
ylabel('V');

figure(2)
v1=[funeval(cl,cdefl,yl,1);funeval(cu,cdefu,yu,1)];
plot(s,v1./s,sstar,v1(N)./sstar,'*');
title('Fish Harvesting: Marginal Value Function');
xlabel('S');
ylabel('V''');
axis([0 1 0 5]);

figure(3)
v2=[funeval(cl,cdefl,yl,2);funeval(cu,cdefu,yu,2)]-v1;
plot(s,v2./(s.^2),sstar,v2(N)./sstar.^2,'*');
title('Fish Harvesting: Curvature of Value Function');
xlabel('S');
ylabel('V"');
axis([0 1 -1 0]);

figure(4)
e=rho*v-(alpha-alpha*s).*v1-(0.5*sigma.^2)*v2;
e=e+(H*(v1-s)).*(s>=sstar);
plot([yl;yu],e)
title('Fish Harvesting: Residual Function');
xlabel('y')
ylabel('e')

prtfigs(mfilename)
```

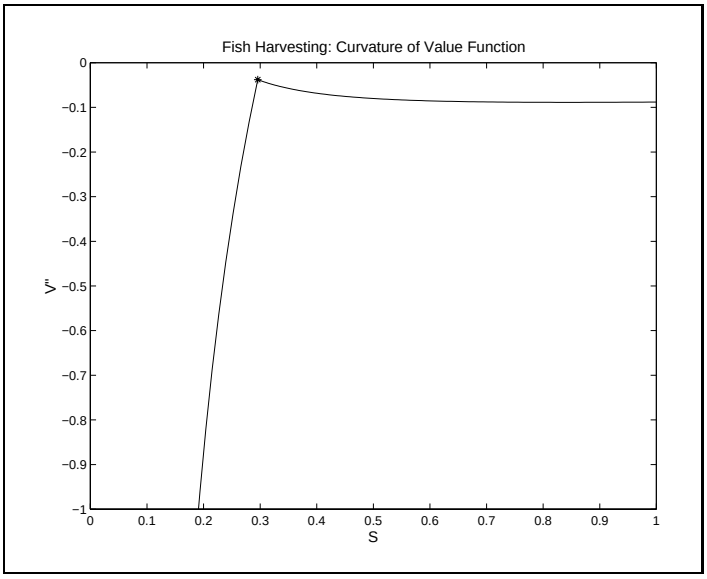


Figure 11.12

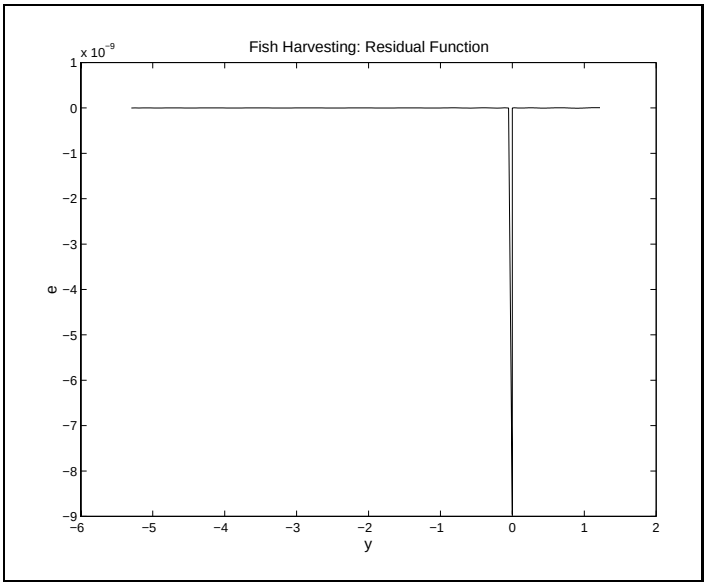


Figure 11.13

in Figure 11.13. One could, of course, approximate the entire value function with, say, a cubic spline, so long as you ensured that y=0 was a node (see problem 7 on

page 454). 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: Entry-Exit**

In stochastic bang-bang problems, the state space is divided into a region in which the control is set to it highest possible value and another region where it is set to its lowest possible value. In other problems with transitional boundaries, the state space may not be partitioned in this way. Instead, multiple free boundaries may need to be determined.

In the entry-exit problem (page 368), a firm is either active or inactive. The two value functions satisfy

$$\begin{array}{rcll} \rho V^a & = & P - C + \mu(P)V^a_P + \sigma^2(P)V^a_{PP} & \text{for } P \in [P_l, \infty) \\ \rho V^i & = & \mu(P)V^i_P + \sigma^2(P)V^i_{PP} & \text{for } P \in [0, P_h] \end{array} . \tag{9}$$

with

$$\begin{aligned} V^i(P_h) &= V^a(P_h) - I \\ V^i(P_l) - E &= V^a(P_l) \\ V^i_P(P_l) &= V^a_P(P_l) \end{aligned}$$

and

$$V^i_P(P_h) = V^a_P(P_h).$$

When $P$ is a geometric Brownian motion process, i.e.,

$$dP = \mu P dt + \sigma P dz,$$

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 $\beta_i$ solve

$$\tfrac{1}{2}\sigma^2\beta(\beta - 1) + \mu\beta - \rho = 0.$$

It can be shown that, for $\rho > 0$, one of the $\beta$ 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 (9)).

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 $A_2^i = 0$. For large $P$, only $V^a$ is defined and the probability of deactivation becomes vanishingly small; hence the value function would approach $P/(\rho-\mu)-C/\rho$, requiring that $A_1^a = 0$.

We still have two unknown constants to determine, $A_1^i$ and $A_2^a$ (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 $P_h$. This is a simple root-finding problem (the toolbox function `entex2.m` solves this problem).

For general processes, we will have to approximate the value functions. We first define two transformed variables that will make the boundaries of the approximating functions simple. Let $y^a = P/P_l$ and $y^i = P/P_h$. The value functions can then be approximated using $V^j(P) = v^j(y^j) \approx \phi^j(y^j)c^j$ for $j = a, i$, with $v^a$ defined on $[1, \bar{y}^a]$ and $v^i$ on $[0, 1]$ (the choice of $\bar{y}^a$, the upper bound on $y^a$ is discussed below).

Given the linearity of the Bellman's Equation and the boundary conditions, it will again be the case that the coefficient vectors $c^i$ and $c^a$, for given values of the boundary points, satisfy a system of linear equations. Our strategy will be write a procedure that is passed trial values of $P^* = [P_l; P_h]$, computes $c^i$ and $c^a$ for these free boundary points and then returns the difference in marginal value functions at the two boundary points:

$$r(P^*) = \begin{bmatrix} -\dfrac{v_y^i(1)}{P_h} + \dfrac{v_y^a(P_h/P_l)}{P_l} \\ \dfrac{v_y^i(P_l/P_h)}{P_h} - \dfrac{v_y^a(1)}{P_l} \end{bmatrix}.$$

At the optimal choice of $P^*$, $r(P^*) = 0$. The procedure that returns the residuals can be passed to a rootfinding algorithm to determine $P^*$.

To see how the coefficient values can be determined, first write the Bellman's equation in terms of the $y^j$ and $v^j$:

$$\rho v^i(y^i) - \frac{\mu(P_h y^i) v^i_y(y^i)}{P_h} - \frac{\sigma^2(P_h y^i) v^i_{yy}(y^i)}{2P_h^2} = 0.$$

and

$$\rho v^a(y^a) - \frac{\mu(P_l y^a) v^a_y(y^a)}{P_l} - \frac{\sigma^2(P_l y^a) v^a_{yy}(y^a)}{2P_l^2} = P_l y^a - C.$$

Suppose the approximating functions have degree $n^i$ and $n^a$, respectively. We will evaluate the Bellman's Equation at $n^i - 2$ values of $y^i$ and $n^a - 2$ values of $y^a$. The addition of the two end point conditions ($v^i(0) = 0$ and $v_{yy}(\bar{y}^a) = 0$) and the two value-matching conditions yields a system of $n^i + n^a$ linear equations in the same number of unknowns.

Specifically, the linear system is

$$\begin{bmatrix} \phi^i(0) & 0 \\ D^i & 0 \\ -\phi^i(1) & (P_h/P_l)\phi^a(P_h/P_l) \\ (P_l/P_h)\phi^i(P_l/P_h) & -\phi^a(1) \\ 0 & D^a \\ 0 & \phi^{a\prime\prime}(\bar{y}^a) \end{bmatrix} \begin{bmatrix} c^i \\ c^a \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ I \\ E \\ -C\underline{1} \\ 0 \end{bmatrix} + P_l \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ Y^a \\ 0 \end{bmatrix},$$

where

$$D^i = \rho\Phi^i_0 - \frac{\mu(P_h Y^i)}{P_h}\Phi^i_1 - \frac{\sigma^2(P_h Y^i)}{2P_h^2}\Phi^i_2$$

and

$$D^a = \rho\Phi^a_0 - \frac{\mu(P_l Y^a)}{P_l}\Phi^a_1 - \frac{\sigma^2(P_l Y^a)}{2P_l^2}\Phi^i_2.$$

The system is of the form $Bc = b_0 + P_l b_1$, where $b_0$ and $b_1$ do not depend on $P_l$ and $P_h$ and hence can be precomputed. Furthermore, all of the basis matrices used in $B$ except for $\phi^i(P_l/P_h)$ and $\phi^a(P_h/P_l)$ can be precomputed. The drift and diffusion terms $\mu$ and $\sigma$ are evaluated at points that may depend on $P_l$ and $P_h$ and hence must be computed each time the residual function is evaluated. The user must supply a specific function to evaluate these terms. Once the drift, diffusion and discount rates are known, the $D^i$ and $D^a$ matrices can be computed and the matrix $B$ can be formed.

The residuals themselves can be written in the form $Rc$:

$$r(P^*) = \begin{bmatrix} -\phi^{i\prime}(1) & \frac{P_h}{P_l}\phi^{a\prime}(P_h/P_l) \\ \frac{P_l}{P_h}\phi^{i\prime}(P_l/P_h) & -\phi^{a\prime}(1) \end{bmatrix} \begin{bmatrix} c^i \\ c^a \end{bmatrix}.$$

The procedure for computing the residuals is found in the toolbox function `entexres`. Most of the inputs to this function are precomputed basis and other parameter matrices, including $b_0$, $b_1$ and elements of $B$ and $R$. A procedure that solves the entry/exit problem is provided in the toolbox function `entex`. This procedure takes as inputs the problem parameters and approximation information. It then precomputes basis matrices and the other coefficients used by the residual function `entexres`, and passes `entexres` to the rootfinding algorithm `broyden` to find $P_l$ and $P_h$. `entex` returns the value of $P^*$, the value function coefficient vectors (`ci` and `ca`) and their associated function definition structures (`fspacei` and `fspacea`).

In addition, a procedure must be defined that accepts values of $P$ (along with additional parameters as needed) and returns values of $\rho$, $\mu$ and $\sigma$ for a specified model.[6] An example of such a file for geometric Brownian motion with a constant discount rate is given by:

```
function [r,m,s]=gbm(P,rho,mu,sigma)
n=size(P,1);
r=rho+zeros(n,1);
m=mu*P;
s=sigma*P;
```

The file `demfb04.m` demonstrates the use of this approach for the geometric Brownian motion case and compares the resulting solution to the (essentially) closed form solution discussed above. Figure 11.14 shows the value functions for the inactive and active states. Figures 11.15 and 11.16 show, respectively, the approximation errors and the residual functions for the collocation approximation relative to the "closed form" solution.

An important point to note about this problem concerns the choice of the upper bound for $v^a$. Even though the optimal boundary values are $P_l = 0.41815$ and $P_h = 2.1996$, it is necessary to set the upper bound for $y^a$ to a rather large number. Intuitively the reason for this stems from the non-stationarity of the geometric Brownian motion process coupled with the infinite time horizon. Together, these imply that the probability of getting a large value of $P$, even when starting at a relatively low current value, is non-negligible over time horizons that contribute to the present value. We should expect, therefore, that relatively larger values of the upper bound will be needed as $\rho - \mu$ shrinks.

[6] Although applications of this model generally treat $\rho$ as a fixed constant, the approach taken here provides the flexibility to make it depend on $P$ if desired.

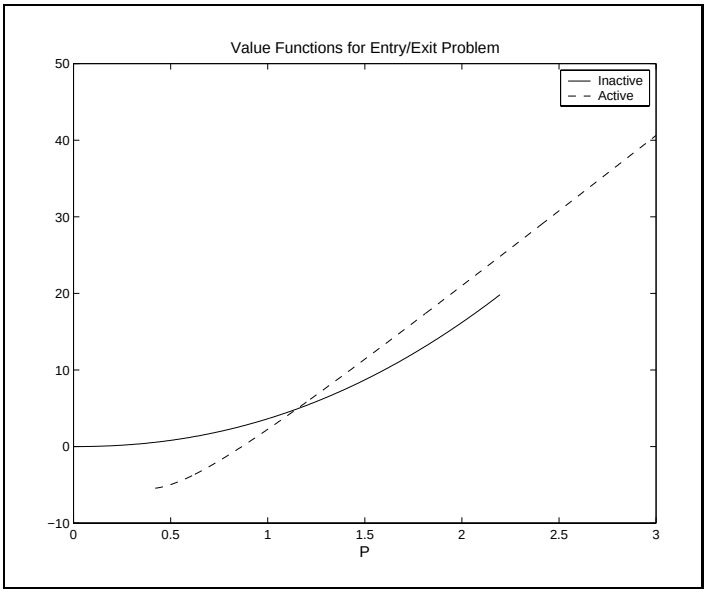


Figure 11.14

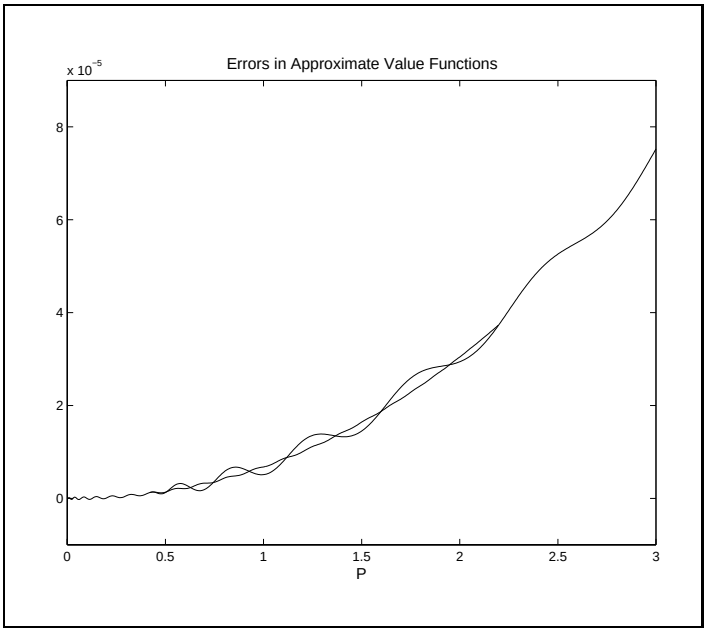


Figure 11.15

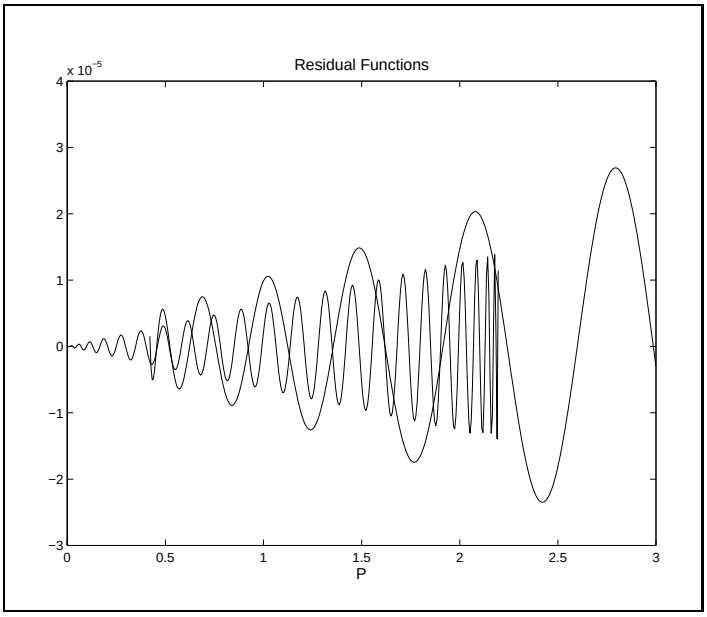


Figure 11.16

### 11.3.2 Finite Horizon Problems

Thus far we have discussed single state, infinite horizon free boundary problems. Somewhat greater difficulties arise in finite horizon problems. The location of the free boundaries in such problems are not isolated points but are functions of time. For example, when an American put option is near to expiration, it is optimal to exercise it at higher prices of the underlying asset than when it is far from expiration.

In finite time problems, we know the value function at the terminal date. This means that we can employ evolutionary methods that work their way backwards in time from the end point. We present two ways of handling such problems. The first implicitly assumes that the control can be exercised only at discrete points in time. Although simple, this method approximates the free boundary with a step function.

To obtain a smoother approximation of the boundary, without requiring a dense set of state variable nodes, we use an explicit finite difference approximation for the time derivative, while simultaneously solving for the free boundary. The two approaches are described in the following two examples.

**Example: Pricing American Options**
In Section 11.1 we solved problems of valuing European style options using the extended method of lines, which approximates the value of an option using $V(S,\tau) \approx$

$\phi(S)c(\tau)$. By evaluating $\phi(S)$ at a set of $n$ nodal values, we derived a differential equation of the form

$$c'(\tau) = \Phi^{-1}Bc(\tau).$$

which is solved by

$$c(\tau + \Delta) = Ac(\tau),$$

where $A = \exp(\Delta\Phi^{-1}B)$ and $\Phi c(0)$ equals the terminal payout, $R(S)$, evaluated at the nodal state values.

The most commonly used strategy for pricing American style options solves the closely related problem of determining the value of an option that can be exercised only at a discrete set of dates. Clearly, as the time between dates shrinks, the value of this option converges to the value of one that can be exercised at any time before expiration.

Between exercise dates, the option is effectively European and hence can be approximated using[7]

$$\hat{c}(\tau + \Delta) = Ac(\tau).$$

The value of $\phi(S)\hat{c}(\tau + \Delta)$ can then be compared to the value of immediate exercise, $R(S)$, and the value function set to the maximum of the two:

$$V(S, \tau + \Delta) \approx \max(R(S), \phi(S)\hat{c}(\tau + \Delta)).$$

The coefficient vector is updated to approximate this function, i.e.,

$$c(\tau + \Delta) = \Phi^{-1} \max(R, \Phi\hat{c}(\tau + \Delta)).$$

The function `finsolve` described in Section 11.1 requires only minor modification to implement this approach to pricing American style assets. First, add an additional field to the model variable, `model.american`, which takes values of 0 or 1. Then, change the main iteration loop to the following:

```
for i=2:N+1
  c(:,i)=A*c(:,i-1);
  if model.american
    c(:,i)=iPhi*max(V0,Phi*c(:,i));
  end
end
```

[7] Commonly used finite difference and binomial tree methods discretize the time derivatives, replacing $c'(\tau + \Delta)$ with $\Big(c(\tau + \Delta) - c(\tau)\Big)/\Delta$.

The file `demfin04.m` demonstrates the use of this feature and produces Figures 11.17-11.19. It closely follows the code described on page 406, but two differences are noteworthy. First, a closed form solution does not exist for the American put option, even when the underlying price is geometric Brownian motion. To assess the quality of the approximation, we have computed a different approximation due to Baron-Adesi and Whaley (see bibliographic notes), which is implemented in the toolbox function `baw`. The differences between the approximations are plotted in Figure 11.18.

The other distinction lies in the determination of the optimal exercise boundary, which is plotted in Figure 11.19. This is obtained by determining which nodal points that are less than or equal to $K$ are associated with an option that is equal to its intrinsic value of $K - S$. The exercise boundary is taken to be the highest such nodal value of $S$. This provides a step function approximation to the early exercise boundary. Unfortunately, this approximation can only be refined by increasing the number of nodal values so they are fairly dense in the region where early exercise may occur (just below the strike price). Such a dense set of nodal values is rarely needed to improve the accuracy of the value function, however.

On the positive side, the method of finding an approximation to the value of an option with a discrete set of exercise dates has two overriding advantages. It is very simple and it extends in an obvious way to multiple state situations. On its negative side, it does not produce a smooth representation of the optimal exercise boundary. If a smooth approximation is needed or desired, the approach described in the next example can be used.

**Example: Sequential Learning**

In the sequential learning problem on page 372, the cumulative production, $Q$, acts like a time variable. There is a known terminal condition at $Q = Q_m$ and the solution can be obtained in an evolutionary fashion by working backwards in $Q$ from $Q_m$. Identifying the location of the free boundary, however, is somewhat more involved than with the American option pricing problem.

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[8]

$$P^*(Q)V_P(P^*(Q), Q) = \beta V(P^*(Q), Q)$$

and

$$P^*(Q)V_{PP}(P^*(Q), Q) = (\beta - 1)V_P(P^*(Q), Q),$$

[8] We ignore the limiting condition as $P \to \infty$.

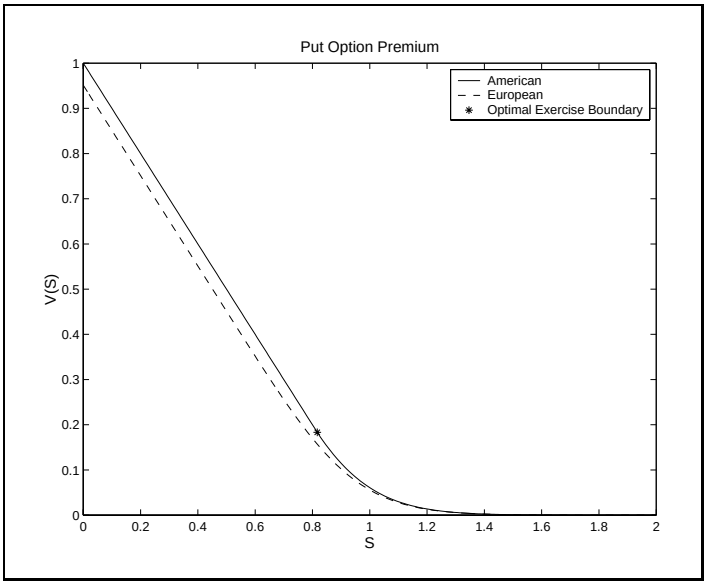


Figure 11.17

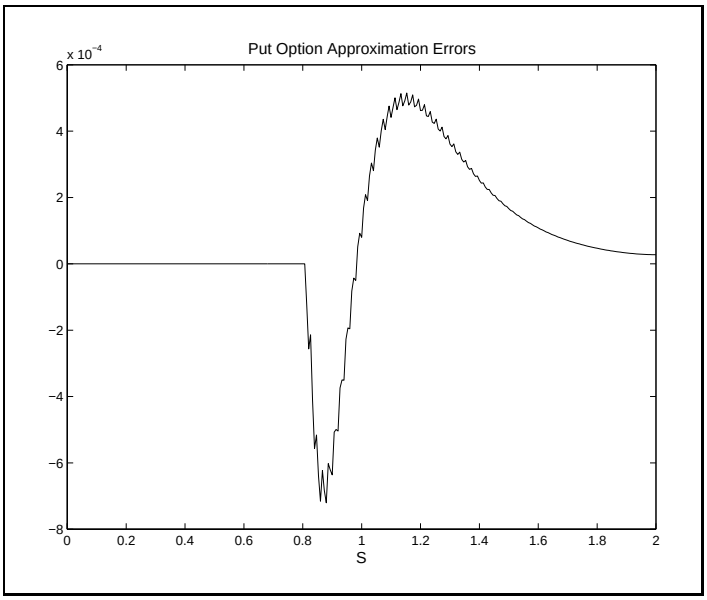


Figure 11.18

where $\beta$ is the positive solution to

$$\tfrac{1}{2}\sigma^2\beta(\beta-1)+(r-\delta)\beta-r=0.$$

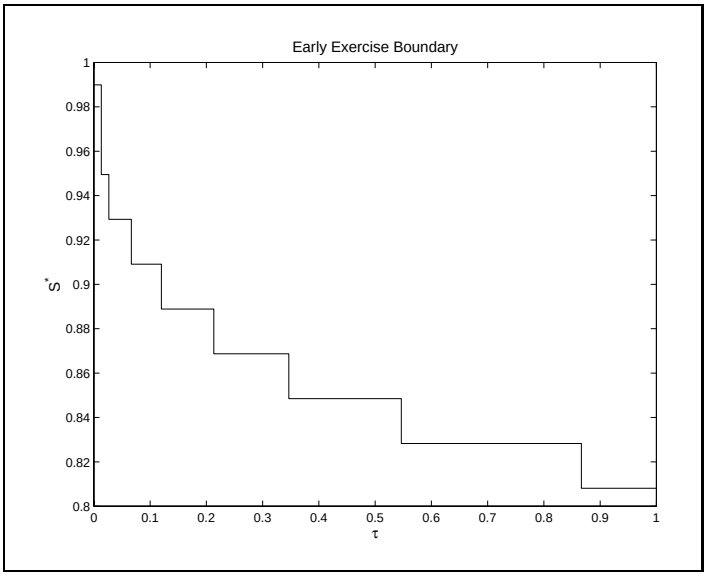


Figure 11.19

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 use a transformation method that regularizes the boundary; specifically,

$$y = \ln(P) - \ln(P^*(Q))$$

with $v(y, Q) = V(P, Q)$. The PDE must be solved for values of $P$ on $[P^*(Q), \infty)$, which translates into values on $y$ on $[0, \infty)$ (in practice we will truncate $y$). Given this transformation it is straightforward to verify the following relationships between the original and the transformed problem:

$$v_y(y, Q) = PV_P(P, Q)$$

$$v_{yy} - v_y = P^2 V_{PP}(P, Q)$$

and

$$V_Q = v_Q - \frac{P^{*\prime}(Q)}{P^*(Q)} v_y.$$

Substituting these expressions into the Bellman equation and the boundary conditions yields:

$$rv = P^* e^y - C(Q) + v_Q + (r - \delta - \tfrac{1}{2}\sigma^2 - P^{*\prime}/P^*)v_y + \tfrac{1}{2}\sigma^2 v_{yy},$$

$$v_y(0,Q) - \beta v(0,Q) = 0$$

and

$$v_{yy}(0,Q) - \beta v_y(0,Q) = 0.$$

We can approximate $v(y,Q)$ with the function $\phi(y)c(Q)$, where $c(Q) : [0, Q_m] \to \Re^n$. The Bellman equation (with suitable rearrangement) can be written as

$$\phi(y)c'(Q) - \frac{\phi'(y)c(Q)}{P^*(Q)}P^{*\prime}(Q) = D(y)c(Q) - e^y P^*(Q) + C(Q),$$

where

$$D(y) = r\phi(y) - (r - \delta - \tfrac{1}{2}\sigma^2)\phi'(y) - \tfrac{1}{2}\sigma^2\phi''(y).$$

The boundary conditions at $y = 0$ ($P = P^*$) are

$$[\phi'(0) - \beta\phi(0)]c(Q) = 0$$

and

$$[\phi''(0) - \beta\phi'(0)]c(Q) = 0.$$

Treated as system of $n+1$ unknowns, this is a differential/algebraic equation (DAE) system in $Q$. It differs from an ordinary system of differential equations because of the boundary conditions, which do not involve the $Q$-derivatives.

We will take a simple explicit Euler approach, by replacing the $Q$ derivatives with first order backwards finite differences

$$c'(Q) \approx \frac{c(Q) - c(Q - \Delta)}{\Delta}$$

and

$$P^{*'}(Q) \approx \frac{P^*(Q) - P^*(Q - \Delta)}{\Delta}.$$

(this approach is explicit because we are working backwards in time from a terminal condition). This leads to the following system

$$\begin{bmatrix} \Phi_0 & -\dfrac{\Phi_1 c(Q)}{P^*(Q)} \\ \phi'(0)-\beta\phi(0) & 0 \\ \phi''(0)-\beta\phi'(0) & 0 \end{bmatrix} \begin{bmatrix} c(Q-\Delta) \\ P^*(Q-\Delta) \end{bmatrix}$$
$$= \begin{bmatrix} \Phi_0 - \Phi_1 - \Delta D(Y) & \Delta e^Y \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} c(Q) \\ P^*(Q) \end{bmatrix} - \begin{bmatrix} C(Q)\underline{1} \\ 0 \\ 0 \end{bmatrix},$$

where $\Phi_0$, $\Phi_1$ and $D$ have the usual meaning, i.e., they are the functions $\phi(y)$, $\phi'(y)$ and $D(y)$ evaluated at $n-1$ nodal points.

A MATLAB function implementing this approach is displayed in Code Box 8. The function takes as input arguments the problem parameters $r$, $\delta$, $\sigma$, $Q_m$, $\bar{c}$, and $C$, as well as the number of $Q$ steps, $N$, and the degree and upper bound of the approximating function, $n$ and `hi`. The function returns an $N+1$ vector of values of $Q$ and an $n \times N+1$ matrix, `c`, of coefficients, each column representing one value of $Q$, with the first column associated with $Q=0$ and the last with $Q=Q_m$. The output `fspace` is the function definition structure associated with $c$. The function also returns the parameters associated with the known parts of the solution, $\beta_1$, $\beta_2$, $A_1$ and $A_2$. A demonstration file `demfb05.m` implements the approach and produces Figure 10.1 on page 374 and Figure 11.20. It uses parameter values of $r=0.05$, $\delta=0.05$, $\sigma=0.02$, $Q_m=20$, $\bar{c}=10$ and $C=40$. The approximation uses the Chebyshev basis of degree $n=15$ for $P$ and 400 evenly spaced steps for $Q$.

A few comments are in order. First, this is effectively an explicit method and hence is prone to instability if the number of nodes and $Q$ steps is not picked carefully. Fortunately, it is generally obvious when this is a problem, as the results will be clearly incorrect. The other problem concerns the choice of the upper bound, $b$. This bound represents a value of $P$ equal to $P^*(Q)e^b$. Too small a value leads to distortions in both the value function and the location of the optimal boundary. By experimenting with this value, we found that having an upper limit of $100P^*$ was sufficient to obtain at least 3 place accuracy for $P^*(Q)$.

**Code Box 11.8: Sequential Learning Problem: Solution Function**

```
% Learn Solves sequential learning problem
function [Q,pstar,c,cdef,A1,A2,beta1,beta2]=learn(r,delta,sigma,Qm,cbar,C,N,n,hi)
% Compute solution for Q>Qm
  beta1=0.5-(r-delta)/sigma.^2 + sqrt((0.5-(r-delta)/sigma.^2).^2+2*r/sigma.^2);
  beta2=0.5-(r-delta)/sigma.^2 - sqrt((0.5-(r-delta)/sigma.^2).^2+2*r/sigma.^2);
% Impose value matching and smooth pasting at P=cbar
  temp=[       cbar.^beta1         -(cbar.^beta2)     ; ...
        beta1*cbar.^(beta1-1)   -beta2*cbar.^(beta2-1)];
  temp=temp\[cbar/delta-cbar/r ; 1/delta];
  A1=temp(1);  A2=temp(2);
% Define the approximating functions and nodal values
  Delta=Qm/N;
  Q=linspace(0,Qm,N+1);
  cdef=fundef({'cheb',n-1,0,hi});
  y=funnode(cdef);
  cdef=fundef({'cheb',n,0,hi});
% Set up collocation matrices
  D=funbasx(cdef,y,[0;1;2]);
  Phi0=D.vals{1};
  Phi1=D.vals{2};
  D=r*Phi0-(r-delta-0.5*sigma^2)*Phi1-0.5*sigma.^2*D.vals{3};
  phi=funbasx(cdef,0,[0;1;2]);
  B=[           Phi0               zeros(n-1,1);
     phi.vals{2}-beta1*phi.vals{1}       0       ;
     phi.vals{3}-(beta1)*phi.vals{2}     0       ];
  A=[Phi0-Phi1-Delta*D  Delta*exp(y);
              zeros(2,n+1)          ];
% Compute cost function values
  gamma=log(C/cbar)/Qm;
  Cost=Delta*cbar*exp(gamma*(Qm-Q));
  Cfactor=[ones(n-1,1);0;0];
% Initialize at terminal boundary
  p=[cbar;cbar*exp(y)];
  c=zeros(n+1,N+1);
  c(1:n,N+1)=[phi.vals{1};Phi0]\(A2*p.^beta2+p/delta-cbar/r);
  c(end,N+1)=cbar;
% Iterate backwards in Q
  for i=N+1:-1:2
    B(1:n-1,end)=Phi1*c(1:n,i)/(-c(end,i));
    c(:,i-1)=B\(A*c(:,i)-Cost(i)*Cfactor);
  end
% Extract Pstar
  pstar=c(end,:)';
  c(end,:)=[];
```

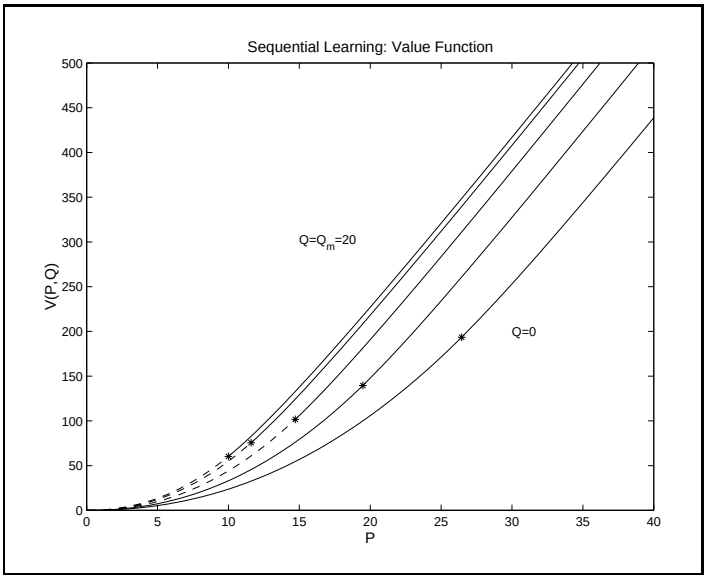
Sequential Learning: Value Function
V(P,Q)
P
Q=Q_m=20
Q=0
0
50
100
150
200
250
300
350
400
450
500
0
5
10
15
20
25
30
35
40

Figure 11.20

**Exercises**

11.1. A Generalized Term Structure Model

A one factor model that encompasses many term-structure models appearing in the literature is based on the process for the short interest rate (under the risk neutral measure) given by:

$$dr = [\alpha_1 + \alpha_2 r + \alpha_3 r \log(r)]dt + [\beta_1 + \beta_2 r]^{\nu} dW.$$

State the PDE satisfied by a zero-coupon bond maturing in $\tau$ periods, along with the associated boundary condition. Write a function analogous to `cirbond` on p. 400 for this case. The function should have the following input/output format:

```
c=Bond(fspace,alpha1,alpha2,alpha3,beta1,beta2,nu,tau)
```

Notice that this function returns the coefficients of a function of $r$ defined by the function definition structure variable `fspace`. Verify that your function reproduces correctly the results obtained using `CIRBOND`, which is a special case of the generalized model. To do this, use the parameters $\tau = 30$, $\kappa = .1$, $\alpha = .05$ and $\sigma = 0.2$.

11.2. Bond Option Pricing

Consider an option that gives its holder the right, in $\tau^o$ periods, to buy a bond that pays 1 unit of account $\tau^b$ periods after the option expires, at a strike price of $K$. Using the model for the short rate described in the previous exercise, write a MATLAB function that computes the value of such an option.The function should have the following input/output format:

```
c=BondOption(fspace,alpha1,alpha2,alpha3,beta1,beta2,nu,K,tauo,taub)
```

Determine the value of an option for the CIR model with $K = 1$, $\tau^o = 1$, $\tau^b = 0.25$, $\kappa = .1$, $\alpha = .05$ and $\sigma = 0.2$.

11.3. Neoclassical Optimal Growth

Use `scsolve` to solve the neoclassical optimal growth model, discussed beginning on page 416:

$$\max_{C(t)} \int_0^{\infty} e^{-\rho t} U(C) dt,$$

subject to the state transition function $K' = q(K) - C$, where

$$q(K) = \alpha \ln(K + 1) - \delta K$$

and

$$U(C) = (C^{1-\gamma} - 1)/(1-\gamma)$$

using the parameter values $\rho = 0.05$, $\delta = 0.02$, $\alpha = 2(\rho+\delta) = 0.14$ and $\gamma = 0.5$. Compare your results to those obtained using the Euler equation approach.

11.4. Cow Replacement

Convert the discrete time and state cow replacement problem on page ?? to a continuous time and continuous state problem and solve the problem.

11.5. Asset Replacement with Stochastic Quality

In the asset replacement problem discussed on pages 359 and 422, the productivity of the asset depended only on its age. Suppose instead that the output of the machine is governed by

$$dQ = -\mu Q dt + \sigma\sqrt{Q(Q-\bar{Q})}dz,$$

where $\bar{Q}$ is the productivity of a new asset. Notice that the process is singular at $Q = 0$ and $Q = \bar{Q}$. At $Q = \bar{Q}$ the drift rate is negative, so productivity is decreasing, whereas $Q = 0$ is an absorbing barrier.

The income flow rate from the asset is $PQ$, for some constant $P$, the replacement cost of the asset is $C$ and the discount rate is $\rho$. Intuitively, there is some value $Q = \beta$ at which it is optimal to replace the asset.

a) State the Bellman's equation and boundary conditions for this problem (be sure to consider what happens if $\beta = 0$). What is the value function for $Q < \beta$?

b) Write a MATLAB file that has the following input/output format:

```
[beta,c,fspace]=Replace(mu,sigma,rho,P,C,Qbar,n)
```

where `c` and `fspace` are the coefficients and the function definition structure defining the value function on the interval $[\beta, \bar{Q}]$, and $n$ is the degree of the approximation. You may assume an interior solution ($\beta > 0$).

c) Call the function you wrote with the line

```
[beta,c,fspace]=Replace(0.02,0.05,0.1,1,2,1,50);
```

Plot the value function on the interval $[0, \bar{Q}]$ (not on $[\beta, \bar{Q}]$) and mark the point $(\beta, V(\beta))$ with a "*".

11.6. Timber Harvesting

a) Solve the timber harvesting example from page 360 with the parameters $\alpha = 0.1$, $m = 1$, $\sigma - 0$, $P = 1$, $C = 0.15$ and $\rho = 0.08$. Plot the value function, indicating the location of the free boundary with an "*".

b) Resolve the problem under the assumption that the land will be abandoned and the replanting cost will not be incurred. Add this result to the plot generated for part (a).

11.7. Fish Harvesting Using Cubic Splines

Modify the code in the fish harvesting example (page 431) to compute the value function using a single cubic spline approximation with a double breakpoint at $y = 0$. Plot the value function and its 1st and 2nd derivatives as functions of $S$ (not $y$) and the residual function for the differential equation as a function of $y$.

11.8. Fish Harvesting - Unbounded Effort

a) Consider the fish harvesting problem (page 431) under the assumption that the control is not bounded ($H \to \infty$), making the problem of the barrier control type. 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 ($\alpha = 0.1$, $\rho = 0.05$, $\sigma = 0.2$).

b) Compute and plot the optimal trigger stock level as a function of the maximal harvest rate ($H$), using the above values for other parameters. Demonstrate that the limiting value as $H \to \infty$ computed in part (a) is correct.

11.9. Cost Uncertainty

Consider 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 $\theta$ .

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

$$\tfrac{1}{2}\nu^2 K F''(K) - F'(K) - 1 = 0.$$

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.

11.10. Investment with Time-to-Build Constraints

Consider an investment project which, upon completion, will have a random value $V$ and will generate a return flow of $\delta V$. The value of the completed project evolves, under the risk neutral measure, according to

$$dV = (r - \delta)Vdt + \sigma V dz,$$

where $r$ is the risk free rate of return. The amount of investment needed to complete the project is $K$, which is a completely controlled process:

$$dK = -Idt,$$

where the investment rate is constrained by $0 \leq I \leq k$. 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

$$
\begin{aligned}
&F(V,0) = V \\
&\lim_{V\to\infty} F_V(V,K) = e^{-\delta K/k} \\
&f(0,K) = 0 \\
&f(V^*,K) = F(V^*,K) \\
&f_V(V^*,K) = F_V(V^*,K).
\end{aligned}
$$

$V^*$ 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

$$\beta = \frac{1}{2} - \frac{r-\delta}{\sigma^2} + \sqrt{\left(\tfrac{1}{2} - \frac{r-\delta}{\sigma^2}\right)^2 + \frac{2r}{\sigma^2}}. \tag{10}$$

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 $\sigma$, $r$, $\delta$ and $k$:

$$\tfrac{1}{2}\sigma^2 V^2 F_{VV} + (r-\delta)VF_V - rF - kF_K - k = 0,$$

subject to

$$
\begin{aligned}
&F(V,0) = V \\
&\lim_{V\to\infty} F_V(V,K) = e^{-\delta K/k} \\
&\beta F(V^*,K) = V^* F_V(V^*,K),
\end{aligned}
$$

where $\beta$ is given by (10). 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 $K$.

The problem as stated is solved by

$$F = Ve^{-\delta/kK} - K$$

with optimal cutoff boundary

$$V^*(K) = \frac{K}{\beta - 1} e^{\delta/kK}.$$

However, the optimal solution must, in addition, satisfy

$$F_K(V^*, K) = 1.$$

Write MATLAB code to solve the time-to-build problem for the following parameter values:

$\delta = 0$
$r = 0.02$
$\sigma = 0.2$
$k = 1$

11.11. Sequential Learning Continued

Review the sequential learning model discussed on pages 372 and 445. Note that the Bellman equation provides an expression for $V_Q$ when $P > P^*$. For $P < P^*$, the value function has the form $A(Q)P^{\beta_1}$ and so $V_Q(P, Q) = A'(Q)P^{\beta_1}$.

a) Derive an expression for $V_Q$ for $P > P^*$.
b) Show that $V_Q$ is continuous at $P = P^*$ for $\sigma > 0$.
c) Use this fact to determine $A'(Q)$.
d) Plot $V_Q$ as a function of $P$ for $Q = 0$ using the parameters $r = \delta = 0.05$ and for the values $\sigma$=0.1, 0.2, 0.3, 0.4 and 0.5.
e) When $\sigma = 0$, $V_Q$ is discontinuous at $P = P^*$. This case was discussed in Problem 20 in Chapter 10. Add this case to the plot from part (d).

# Appendix A
# Basis Matrices for Multivariate Models

The basic PDE used in control problems arising in economics and finance takes the form

$$r(S)V = \delta(S) + V_t + V_{SS}\mu(S) + \tfrac{1}{2}\text{trace}(\sigma(S)\sigma(S)^\top V_{SS}).$$

Generically, $S$ is a $d$-dimensional vector, $V_S$ is $(1 \times d)$, $\mu(S)$ is $d \times 1$, $V_{SS}$ is $d \times d$, and $\sigma(S)$ is $d \times d$.

The first derivative term can be computed (for a single $S$ vector) using

$$V_S\mu(S) \approx \mu(S)^\top \phi'(S)c$$

where

$$\phi'(S) = \begin{bmatrix} \phi_1'(S_1) \otimes \phi_2(S_2) \otimes \ldots \otimes \phi_d(S_d) \\ \phi_1(S_1) \otimes \phi_2'(S_2) \otimes \ldots \otimes \phi_d(S_d) \\ \ldots \\ \phi_1(S_1) \otimes \phi_2(S_2) \otimes \ldots \otimes \phi_d'(S_d) \end{bmatrix}.$$

The second derivative term can be computed using

$$\tfrac{1}{2}\text{trace}(\sigma(S)\sigma(S)^\top V_{SS}) \approx \tfrac{1}{2}\text{vec}(\sigma(S)\sigma(S)^\top)^\top \phi''(S)c$$

where

$$\phi''(S) = \begin{bmatrix} \phi_1''(S_1) \otimes \phi_2(S_2) \otimes \ldots \otimes \phi_d(S_d) \\ \phi_1'(S_1) \otimes \phi_2'(S_2) \otimes \ldots \otimes \phi_d(S_d) \\ \ldots \\ \phi_1'(S_1) \otimes \phi_2(S_2) \otimes \ldots \otimes \phi_d'(S_d) \\ \phi_1'(S_1) \otimes \phi_2'(S_2) \otimes \ldots \otimes \phi_d(S_d) \\ \phi_1(S_1) \otimes \phi_2''(S_2) \otimes \ldots \otimes \phi_d(S_d) \\ \ldots \\ \phi_1(S_1) \otimes \phi_2'(S_2) \otimes \ldots \otimes \phi_d'(S_d) \\ \ldots \\ \phi_1'(S_1) \otimes \phi_2(S_2) \otimes \ldots \otimes \phi_d'(S_d) \\ \phi_1(S_1) \otimes \phi_2'(S_2) \otimes \ldots \otimes \phi_d'(S_d) \\ \ldots \\ \phi_1(S_1) \otimes \phi_2(S_2) \otimes \ldots \otimes \phi_d''(S_d) \end{bmatrix}.$$

It would be even more efficient to avoid the double computations arising from symmetry by using the vech operator:

$$\tfrac{1}{2}\text{trace}(\sigma(S)\sigma(S)^\top V_{SS}) \approx \text{vech}(\sigma(S)\sigma(S)^\top)^\top \hat{\phi}''(S)c$$

where

$$\hat{\phi}''(S) = \begin{bmatrix} \frac{1}{2}\phi_1''(S_1) \otimes \phi_2(S_2) \otimes \ldots \otimes \phi_d(S_d) \\ \phi_1'(S_1) \otimes \phi_2'(S_2) \otimes \ldots \otimes \phi_d(S_d) \\ \ldots \\ \phi_1'(S_1) \otimes \phi_2(S_2) \otimes \ldots \otimes \phi_d'(S_d) \\ \phi_1(S_1) \otimes \frac{1}{2}\phi_2''(S_2) \otimes \ldots \otimes \phi_d(S_d) \\ \ldots \\ \phi_1(S_1) \otimes \phi_2'(S_2) \otimes \ldots \otimes \phi_d'(S_d) \\ \ldots \\ \phi_1(S_1) \otimes \phi_2(S_2) \otimes \ldots \otimes \frac{1}{2}\phi_d''(S_d) \end{bmatrix}.$$

## Bibliographic Notes

A standard reference on solving PDEs is Ames. It contains a good discussion of stability and convergence analysis; the section on parabolic PDEs is especially relevant for economic applications. Golub and Ortega contains a useful introductory treatment the extended method-of-lines for solving PDEs (Section 8.4), which they call a semi-discrete method. Most treatments of PDEs begin with a discussion of finite difference methods and may then proceed to finite element and weighted residual methods. The approach we have taken reverses this order by starting with a weighted residual approach (collocation) and demonstrating that finite difference methods can be viewed as a special case with a specific choice of basis functions. We have not discussed finite element methods explicitly, but the same remarks apply to them. Piecewise linear cubic splines bases are common examples of finite element methods.

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.

Numerous references containing discussions of numerical techniques for solving financial asset models now exist. Hull contains a good overview of commonly used techniques. See also Duffie and Wilmott. In addition to finite difference methods, binomial and trinomial trees and Monte Carlo methods are the most commonly used approaches.

Tree approaches represent state dynamics using a branching process. Although the conceptual framework seems different from the PDE approach, tree methods are computationally closely related to explicit finite difference methods for solving PDEs. If the solution to an asset pricing model for a given initial value of the state is the only output required from a solution, trees have an advantage over finite difference methods because they require evaluation of far fewer nodal points. If the entire solution function and/or derivatives with respect to the state variable and to time are desired, this advantage disappears. Furthermore, the extended method of lines is quite competitive with tree methods and far more simple to implement.

Monte Carlo techniques are increasingly being used, especially in situations with a high dimensional state space. The essential approach simulates paths for the state variable using the risk-neutral state process. Many assets can then be priced as the average value of the returns to the asset evaluated along each sample path. This approach is both simple to implement and avoids the need for special treatment of boundary conditions with exotic assets. Numerous refinements exist to increase the efficiency of the approach, including the use of variance reduction techniques such as antithetic and control variates, as well as the use of quasi-random numbers (low discrepancy sequences). Monte Carlo approaches have been applied to the calculation of American style assets with early exercise features but this requires more work.

Other approaches to solving stochastic control problems include discretization methods; see, e.g., Kushner and Dupuis.

Several of the exercises are based on problems in the literature. The generalized model of the short interest rate appears in Duffie, pp. 131-133. The fish harvesting problem with adjustment costs was developed by Ludwig and Ludwig and Varrah. The cost uncertainty model is discussed in Dixit and Pindyck, pp. 345-351. The time-to-build exercise is from Majd and Pindyck and is also discussed in Dixit and Pindyck (pp. 328-339).

# 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 $\alpha f$ 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, $x \cdot y$, is the real number $\sum_i x_i y_i$. And their array product, $z = x. * y$, is the $n$-vector whose $i^{th}$ entry is $z_i = x_i y_i$.

If $\alpha$ 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, $z = \alpha x = x\alpha$, is the $n$-vector whose $i^{th}$ entry is $z_i = \alpha x_i$.

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:

$$\begin{aligned} ||x||_1 &= \textstyle\sum_i |x_i|, \\ ||x||_2 &= \sqrt{\textstyle\sum_i |x_i|^2}, \\ ||x||_\infty &= \max_i |x_i|. \end{aligned}$$

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 $C = AB$ 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 $C = A. * B$ 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 $A^\top A = AA^\top$ is diagonal, and orthonormal if $A^\top A = AA^\top = I$. 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 $AA^{-1} = A^{-1}A = I$. 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

$$||A|| = \max_{||x||=1} ||Ax||$$

or, equivalently,

$$||A|| = \max_{||x|| \neq 0} \frac{||Ax||}{||x||}.$$

Given corresponding vector and matrix norms,

$$||Ax|| \leq ||A||\ ||x||.$$

Moreover, if $A$ and $B$ are square matrices,

$$||AB|| \leq ||A||\ ||B||.$$

Common matrix norms include the matrix norms induced by the one, two (Euclidean), and infinity norms:

$$||A||_p = \max_{||x||_p=1} ||Ax||_p$$

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 two (Euclidean) matrix norm is relatively expensive to compute. The one and infinity norms, on the other hand, take a relatively simple form:

$$\begin{array}{rcl} ||A||_1 & = & \max_{1\leq j\leq n} \sum_{i=1}^{n} |A_{ij}| \\ ||A||_\infty & = & \max_{1\leq i\leq n} |A_{ij}|. \end{array}$$

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 $x^\top Ax \leq 0$ for all $x$; it is negative definite if $x^\top Ax < 0$ for all $x \neq 0$; it is positive semidefinite if $x^\top Ax \geq 0$ for all $x$; and it is positive definite if $x^\top Ax > 0$ 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 $S$,

$$f(x) = f(x_0) + f_x(x_0)(x - x_0) + o(||x - x_0||)$$

and

$$f(x) = f(x_0) + f_x(x_0)(x - x_0) + \tfrac{1}{2}(x - x_0)^\top f_{xx}(x_0)(x - x_0) + o(||x - x_0||^2)$$

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 $f'(x) = -F_y^{-1}(x, f(x))F_x(x, f(x))$.

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 $\pi$, 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

$$\begin{aligned} P^\top \pi &= \pi \\ \textstyle\sum_i \pi_i &= 1, \end{aligned}$$

then the Markov chain has a steady-state distribution $\pi$.

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

$$\begin{bmatrix} I - P^\top \\ \underline{1}^\top \end{bmatrix} \pi = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

where $P$ is the probability transition matrix and $\underline{1}$ 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 $\pi$ of the Markov chain may be computed by solving the linear equation

$$\begin{bmatrix} 0.5 & 0.0 & -0.5 \\ -0.2 & 0.6 & -0.5 \\ 1.0 & 1.0 & 1.0 \end{bmatrix} \pi = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

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.

## A.5 Continuous Time Mathematics

### A.5.1 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 $\Delta t$.

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

$$dS = \mu(S,t)dt + \sigma(S,t)dz$$

where $z$ is a standard Wiener process. The Ito process in completely defined in terms of the functions $\mu$ and $\sigma$, which can be interpreted as the instantaneous mean and standard deviation of the process:

$$E[dS] = \mu(S,t)dt$$

and

$$Var[dS] = E[dS^2] - (E[dS])^2 = E[dS^2] - \mu(S,t)^2 dt^2 = E[dS^2] = \sigma^2(S,t)dt,$$

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 definition of the two functions.

The differential representation is a shorthand for the stochastic integral

$$S_{t+\Delta t} = S_t + \int_t^{t+\Delta t} \mu(S_\tau, \tau)d\tau + \int_t^{t+\Delta t} \sigma(S_\tau, \tau)dz. \tag{1}$$

The first of the integrals in (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:

$$\int_t^{t+\Delta t} \sigma(S_\tau, \tau)dz = \lim_{n\to\infty} \sqrt{\frac{\Delta t}{n}} \sum_{i=0}^{n-1} \sigma(S_{t+ih}, t+ih)v_i, \tag{2}$$

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 $S$ that are not yet realized are not used to evaluate the $\sigma$ function. This naturally represents the notion that current events cannot be functions of specific realizations of future events.[1] It is useful to note that $E_t dS = \mu(S,t)dt$; this is a direct consequence of the fact that each of the elements of the sum in (2) has zero expectation. This implies that

$$E_t[S_{t+\Delta t}] = S_t + E_t \int_t^{t+\Delta t} \mu(S_\tau, \tau) d\tau.$$

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

$$S_{t+\Delta t} = S_t + \mu(S_t, t)\Delta t + \sigma(S_t, t)\sqrt{\Delta t}\, v,$$

where $v \sim N(0,1)$. This approximation will be exact when $\mu$ and $\sigma$ are constants.[2] In other cases the approximation will improve as $\Delta t$ gets small, but may produce inaccurate results as $\Delta t$ gets large.

In order to define and work with functions of Ito processes it is necessary to have a calculus that operates consistently with them. Suppose $y = f(t, S)$, with continuous derivatives $f_t$, $f_S$ and $f_{SS}$. In the simplest case $S$ and $y$ are both scalar processes. It is intuitively reasonable to define the differential $dy$ as

$$dy = f_t dt + f_S dS,$$

as would be the case in standard calculus. Unfortunately, this will produce incorrect results because it ignores the fact that $(dS)^2 = O(dt)$. To see what this means consider

[1] 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 $\lambda$ is any value on $[0,1]$. With stochastic integrals, alternative values on $\lambda$ produce different results. Furthermore, any value of $\lambda$ other than 0 would imply a sort of clairvoyance that makes it unsuitable for applications involving decision making under uncertainty.

[2] When $\mu$ and $\sigma$ are constants the process is known as absolute Brownian motion. Exact simulation methods also exist for other processes, e.g., for geometric Brownian motion process,

$$dS = \mu S dt + \sigma S dz,$$

it will subsequently be shown that

$$S_{t+\Delta t} = S_t \exp(\mu\Delta t + \sigma\sqrt{\Delta t} v),$$

where $v \sim N(0,1)$.

a Taylor expansion of $dy$ at $(S,t)$, i.e., totally differentiate the Taylor expansion of $f(S,t)$:

$$dy = f_t dt + f_S dS + \tfrac{1}{2} f_{tt}(dt)^2 + f_{tS} dt dS + \tfrac{1}{2} f_{SS}(dS)^2 + \text{ higher order terms.}$$

Terms of higher order than $dt$ and $dS$ are then ignored in the differential. In this case, however, the term $(dS)^2$ represents the square of the increments of a random variable that has expectation $\sigma^2 dt$ and, therefore, cannot be ignored. Including this term results in the differential

$$\begin{aligned} dy &= [f_t + \tfrac{1}{2} f_{SS}\sigma^2(S,t)]dt + f_S dS \\ &= [f_t + f_S\mu(S,t) + \tfrac{1}{2} f_{SS}\sigma^2(S,t)]dt + f_S\sigma(S,t)dz, \end{aligned}$$

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 appropriately continuous derivatives).

Multivariate versions of Ito's Lemma are easily defined. Suppose $S$ is an $n$-vector valued process and $z$ is a $k$-vector Wiener process (composed of $k$ independent standard Wiener processes). Then $\mu$ is an $n$-vector valued function ($\mu : \Re^{n+1} \to \Re^n$) and $\sigma$ is an $n \times k$ matrix valued function ($\sigma : \Re^{n+1} \to \Re^{n\times k}$). The instantaneous covariance of $S$ is $\sigma\sigma^T$, which may be less than full rank.

For vector-valued $S$, Ito's Lemma is

$$dy = \left[f_t + f_S\mu(S,t) + \tfrac{1}{2}\text{trace}\left(\sigma^T(S,t) f_{SS}\sigma(S,t)\right)\right]dt + f_S\sigma(S,t)dz,$$

(the only difference being in the second order term; derivatives are defined such that $f_S$ is a $(1 \times n)$-vector). The lemma extends in an obvious way if $y$ is vector valued.

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

$$dS = \mu S dt + \sigma S dz.$$

Define $y = \ln(S)$, implying that $\partial y/\partial t = 0$, $\partial y/\partial S = 1/S$ and $\partial^2 y/\partial S^2 = -1/S^2$. Applying Ito's Lemma yields the result that

$$dy = [\mu - \sigma^2/2]dt + \sigma dz.$$

This is a process with independent increments, $y_{t+\Delta t} - y_t$, 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:

$$\ln(S_{t+\Delta t}) - \ln(S_t) \sim N\left((\mu - \sigma^2/2)\Delta t, \sigma^2\Delta t\right).$$

### A.5.2 Forward and Backward Equations

It is often useful to consider the behavior of a process at some future time, $T$, from the vantage point of the current time, $t$. Suppose, for example, we are interested in deriving an expression for $E[S_T|S_t = s] = m(s,t,T)$, where $dS_t = \mu dt + \sigma dz$. Notice that there are two time variables in this function, $T$ and $t$. It is natural, therefore, that the behavior of the function can be expressed in terms of differential equations in either of these variables. When $T$ is held fixed and $t$ varies, the resulting differential equation is a "backward" equation; when $t$ is held fixed and $T$ varies, it is a "forward" equation.

The forward approach uses the integral representation of the SDE

$$S_T = S_t + \int_t^T \mu(S_\tau, \tau)d\tau + \int_t^T \sigma(S_\tau, \tau)dz_\tau.$$

The diffusion term has expectation 0, so

$$\mathrm{E}_t[S_T] = S_t + \int_t^T \mathrm{E}_t\left[\mu(S_\tau, \tau)\right] d\tau$$

or, in differential form,

$$\frac{\partial \mathrm{E}_t[S_T]}{\partial T} = \mathrm{E}_t\left[\mu(S_T, T)\right]. \tag{3}$$

If $\mu$ is affine in $S$, $\mu(S) = \kappa(\alpha - S)$, this leads to the differential equation $m_T = \kappa(\alpha - m)$, with the boundary condition at time $t$ that $m(s,t,t) = s$. Thus

$$E[S_T|S_t = s] = \alpha + e^{-\kappa(T-t)}(s - \alpha). \tag{4}$$

In contrast, the backward approach holds $T$ fixed. Viewing $m$ as a process that varies in $t$ and using Ito's Lemma

$$dm = \left[m_t + m_S\mu + \frac{1}{2}m_{SS}\sigma^2\right] dt + m_S\sigma dz. \tag{5}$$

By the Law of Iterated Expectations, the drift associated with the process $m$ must be 0; hence $m$ solves the partial differential equation

$$0 = m_t + m_S\mu + \frac{1}{2}m_{SS}\sigma^2,$$

subject to the boundary condition that $m(s,T,T) = s$. For the affine $\mu$, the differential equation is

$$0 = m_t + m_S\kappa(\alpha - S) + \frac{1}{2}m_{SS}\sigma^2(S_t, t).$$

Although the $\sigma$ term appears in this PDE, it actually plays no role. We leave as an exercise the verification that this PDE is solved by the function obtained from the forward equation.

Forward and backwards equations can also be derived for expectations of functions of $S$. Consider the function $S_t^2$; Ito's Lemma provides an expression for its dynamics:

$$S_T^2 = S_t^2 + \int_t^T S_\tau \mu(S_\tau, \tau) + \sigma^2(Sx_\tau, \tau) d\tau + \int_t^T S_\tau \sigma(S_\tau, \tau) dz.$$

Taking expectations and subtracting the square of $E_t[S_T]$ provides an expression for the variance of $S_T$ given $S_t$:

$$Var_t[S_T] = S_t^2 + E_t\left[\int_t^T S_\tau \mu(S_\tau, \tau) + \sigma^2(S_\tau, \tau) d\tau\right] - \left(E_t[S_T]\right)^2.$$

Differentiating this with respect to $T$ yields

$$\frac{dVar_t[S_T]}{dT} = E_t[\sigma^2(S_T, T)] + 2\Big(E_t[S_T \mu(S_T, T)] - E_t[S_T] E_t[\mu(S_T, T)]\Big).$$

The boundary condition is that $Var_T[S_T] = 0$, i.e., at time $T$ all uncertainty about the value of $S_T$ is resolved. As an exercise, you are asked to apply this result to the process

$$dS = \kappa(\alpha - S)dt + \sigma dz$$

(i.e., the diffusion term is a constant).

The backward approach can also be used. Consider again the expression (5), noting that the drift equals 0, so

$$dm = m_s(S_t, t, T)\sigma(S_t, t) dz.$$

Furthermore, $m_T = S_T$, so

$$S_T = m_t + \int_t^T m_s(S_\tau, \tau, T)\sigma(S_\tau, \tau) dz_\tau,$$

the variance of which is

$$Var_t[S_T] = \mathrm{E}_t[(S_T - m_t)^2] = \mathrm{E}_t\left[\left(\int_t^T m_s \sigma dz_\tau\right)^2\right].$$

Given two functions $f(S_t, t)$ and $g(S_t, t)$, it can be shown that

$$\begin{aligned}\mathrm{E}_t\left[f(S_T, T) g(S_T, T)\right] &= \mathrm{E}_t\left[\left[\int_t^T f(S_\tau, \tau) dW_\tau\right]\left[\int_t^T g(S_\tau, \tau) dW_\tau\right]\right] \\ &= \mathrm{E}_t\left[\int_t^T f(S_\tau, \tau) g(S_\tau, \tau) d\tau\right]\end{aligned}$$

and therefore

$$Var_t[S_T] = \mathrm{E}_t\left[\int_t^T m_s^2(S_\tau,\tau,T)\sigma^2(S_\tau,\tau)d\tau\right]. \tag{6}$$

Another important use of forward and backward equations is in providing expressions for the transition densities associated with stochastic processes. Let $f(S,T;s,t)$ denote the density function defined by

$$Prob[S_T \le S|S_t = s] = \int_{-\infty}^{S} f(S_T,T;s,t)dS_T.$$

The Kolmogorov forward and backward equations are partial differential equations satisfied by $f$. The forward equation, which treats $S$ and $T$ as variable, is

$$0 = \frac{\partial f(S,T;s,t)}{\partial T} + \frac{\partial \mu(S,T)f(S,T;s,t)}{\partial S} - \tfrac{1}{2}\frac{\partial^2\sigma^2(S,T)f(S,T;s,t)}{\partial S^2}.$$

From the definition of the transition density function, $f$ must have a degenerate distribution at $T=t$, i.e.,

$$f(S,t;s,t) = \delta_s(S),$$

where $\delta_s(S)$ is the Dirac function which concentrates all probability mass at the single point $S=s$.

Similarly, the backward equation, which treats $s$ and $t$ as variable, is

$$0 = \frac{\partial f(S,T;s,t)}{\partial t} + \mu(s,t)\frac{\partial f(S,T;s,t)}{\partial s} + \tfrac{1}{2}\sigma^2(s,t)\frac{\partial^2 f(S,T;s,t)}{\partial s^2}.$$

The boundary condition for the backward equation is the terminal condition
$f(S,T;s,T) = \delta_S(s)$.

We leave as an exercise the verification that

$$dS = \kappa(\alpha - S)dt + \sigma dz$$

has Gaussian transition densities, i.e., that

$$f(S,T;s,t) = \frac{1}{\sqrt{2\pi v}}\exp(-0.5(S-m)^2/v),$$

where $m$ is given in (4) and

$$v = \frac{\sigma^2}{2\alpha}\left(1 - e^{-2\kappa(T-t)}\right).$$

### A.5.3 The Feynman-Kac Equation

The backward equation approach to computing moments is a special case of a more general result on the relationship between the solution to certain PDEs and the expectation of functions of diffusion processes. Control theory in continuous time is typically concerned with problems which attempt to choose a control that maximizes an expected 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

$$\rho V(S,t) = f(S) + V_t(S,t) + \mu(S)V_S(S,t) + \tfrac{1}{2}\sigma^2(S)V_{SS}(S,t),$$

with $V(S,T) = R(S)$. The function $R$ here represents a terminal value of the state, i.e., a salvage value.[3]

By applying Ito's Lemma, the Feynman-Kac Equation can be expressed as:

$$\rho V(S,t) = f(S) + E[dV]/dt. \tag{7}$$

(7) 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, $\rho V$, 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

$$V(S_t) = E_t \left[ \int_t^\infty e^{-\rho(\tau - t)} f(S) d\tau \right]$$

is the solution to the differential equation

$$\rho V(S) = f(S) + \mu(S)V_S(S) + \tfrac{1}{2}\sigma^2(S)V_{SS}(S).$$

[3]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 10.3.3).

Although more general versions of the theorem exist (see bibliographical notes), 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 $\infty$ in economic problems). Generally economic intuition about the nature of the problem is used to determine the boundary conditions.

**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

$$dS = \mu S dt + \sigma S dz,$$

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 $\beta_i$ 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 $\rho$, one of these roots is greater than one, the other is negative: $\beta_1 > 1$, $\beta_2 < 0$.

Consider the problem of finding the expected discounted value of a power of $S$, ($f(S) = S^\gamma$), assuming, momentarily, that the expectation exists. It is easily verified that a particular solution is

$$V(S) = S^\gamma/(\rho - \mu\gamma - \tfrac{1}{2}\sigma^2\gamma(\gamma - 1)). \tag{8}$$

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 exists (the integral may not converge as its upper limit of integration goes to $\infty$). It can be shown, however, that the present value is well defined for $\beta_2 < \gamma < \beta_1$, making the numerator in (8) 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.

## Bibliographic Notes

Many books contain discussions of Ito stochastic calculus with economics and finance orientation, including Neftci and Hull. 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 (1975). For more detail, including discussion of boundary issues, see Karlin and Taylor, chapter 15 and Bharucha-Reid. Early work in this area is contained in several papers by Feller. A classic text on stochastic processes is Cox and Miller.

# Appendix B

# A MATLAB Primer

## B.1 The Basics

MATLAB is a programming language and a computing environment that uses matrices as one of its basic data types. It is a commercial product developed and distributed by MathWorks. Because it is a high level language for numerical analysis, numerical code to be written very compactly. For example, suppose you have defined two matrices (more on how to do that presently) that you call $A$ and $B$ and you want to multiply them together to form a new matrix $C$. This is done with the code

```
C=A*B;
```

(note that expressions generally terminate with a semicolon in MATLAB). In addition to multiplication, most standard matrix operations are coded in the natural way for anyone trained in basic matrix algebra. Thus the following can be used

`A+B`
`A-B`
`A'` for the transpose of $A$
`inv(A)` for the inverse of $A$
`det(A)` for determinant of $A$
`diag(A)` for a vector equal to the diagonal elements of $A$

With the exception of transposition all of these must be used with appropriate sized matrices, e.g., square matrices to `inv` and `det` and conformable matrices for arithmetic operations. In addition, standard mathematical operators and functions are defined that operate on each element of a matrix. For example, suppose $A$ is defined as the $2 \times 1$ matrix

```
[2 3]
```

then `A.^2` (`.^` is the exponentiation operator) yields

```
[4 9]
```

(not `A*A`, which is not defined for non-square matrices anyway). Functions that operate on each element include

`exp`
`ln`
`sqrt`
`cos`
`sin`
`tan`
`arccos`
`arcsin`
`arctan`
and
`abs`

In addition to these standard mathematical functions there are a number of less standard but useful functions such as cumulative distribution functions for the normal: `cdfn` (in the STATS toolbox). The constant $\pi$ (`pi`) is also available. MATLAB has a large number of built-in functions, far more than can be discussed here.

As you explore the capabilities of MATLAB a useful tool is MATLAB's help documentation. Try typing `helpwin` at the command prompt; this will open a graphical interface window that will let you explore the various type of functions available. You can also type `help` or `helpwin` followed by a specific command or function name at the command prompt to get help on a specific topic.

Be aware that MATLAB can only find a function if it is either a built-in function or is in a file that is located in a directory specified by the MATLAB path. If you get a function or variable not found message, you should check the MATLAB path (using `path` to see if the functions directory is included) or use the command `addpath` to add a directory to the MATLAB path. Also be aware that files with the same name can cause problems. If the MATLAB path has two directories with files called `tiptop.m`, and you try to use the function `tiptop`, you may not get the function you want. You can determine which is being used with the which command, e.g., `which tiptop`, and the full path to the file where the function is contained will be displayed.

A few other built in functions or operators are extremely useful, especially

```
index=start:increment:end;
```

creates a row vector of evenly spaced values. For example,

```
i=1:1:10;
```

creates the vector [1 2 3 4 5 6 7 8 9 10]. It is important to keep track of the dimensions of matrices; the size function does this. For example, if $A$ is $3 \times 2$,

```
size(A,1)
```

returns a 3 and

```
size(A,2)
```

returns a 2. The second argument of the size function is the dimension: the first dimension of a matrix is the rows, the second is the columns. If the dimension is left out a $1 \times 2$ vector is returned:

```
size(A)
```

returns [3 2]. There are a number of ways to create matrices. One is by enumeration

```
X=[1 5;2 1];
```

which defines $X$ to be the $2 \times 2$ matrix

$$\begin{bmatrix} 1 & 5 \\ 2 & 1 \end{bmatrix}$$

The ; indicates the end of a row (actually it is a concatenation operator that allow you to stack matrices; more on that below). Other ways to create matrices include

```
X=ones(m,n);
```

and

```
X=zeros(m,n);
```

which create $m \times n$ matrices with each element equal to 1 or 0, respectively. MATLAB also has several random number generators with a similar syntax.

```
X=rand(m,n);
```

creates an $m \times n$ matrix of independent random draws from a uniform distribution (actually they are pseudo-random).

```
X=randn(m,n);
```

draws from the standard normal distribution. Individual elements of a matrix the size of which has been defined can be accessed using `()`; for example if you have defined the $3 \times 2$ matrix $B$, you can set element 1,2 equal to cos(2.5) with the statement

```
B(1,2)=cos(5.5);
```

If you then what to set element 2,1 to the same value use

```
B[2,1]=B[1,2];
```

A whole column or row of a matrix can be referenced as well in the following way

```
B(:,1);
```

refers to column 1 of the matrix $B$ and

```
B(3,:);
```

refers to its third row. The : is an operator that selects all of the elements in the row or column. An equivalent expression is

```
B(3,1:end);
```

where end indicates the column in the matrix. You can also pick and choose the elements you want, e.g.,

```
C=B([1 3],2);
```

results in a new $2 \times 1$ matrix equal to

$$\begin{bmatrix} B_{12} \\ B_{32} \end{bmatrix}.$$

Also the construction

```
B(1:3,2);
```

is used to refer to rows 1 through 3 and column 2 of the matrix $B$.

The ability to access parts of a matrix is very useful but also can cause problems. One of the most common programming errors is attempting to access elements of a matrix that don't exist; this will cause an error message. While on the subject on indexing elements of a matrix, you should know that MATLAB actually has two different ways of indexing. One is to use the row and column indices, as above, the other to use the location in the vectorized matrix. When you vectorize a matrix you stack its columns on top of each other. So a $3 \times 2$ matrix becomes a $6 \times 1$ vector composed of a stack of two $3 \times 1$ vectors. Element 1,2 of the matrix is element 4 of the vectorized matrix. If you want to create a vectorized matrix the command

```
X(:)
```

will do the trick.

MATLAB has a powerful set of graphics routines that enable you to visualize your data and models. For starters, it will suffice to note that routines plot, mesh and contour. For plotting in two dimensions, use `plot(x,y)`. Passing a string as a third argument gives you control over the color of the plot and the type of line or symbol used. `mesh(x,y,z)` provides plots of a 3-D surface, whereas `contour(x,y,z)` projects a 3-d surface onto two dimensions. It is easy to add titles, labels and text to the plots using `title`, `xlabel`, `ylabel` and `text`. Subscripts, superscripts and Greek letters can be obtained using TEX commands (e.g., `x_t`, `x^2` and `\alpha\mu` will result in $x_t$, $x^2$ and $\alpha\mu$).

To gain mastery over graphics takes some time; the documentation Using MATLAB Graphics available with MATLAB is as good a place as any to learn more. You may have noticed that statements sometimes end with ; (semi-colon) and they don't. MATLAB is an interactive environment, meaning it interacts with you as it runs jobs. It communicates things to you via your display terminal. Any time MATLAB executes an assignment statement, meaning that is assigns new values to variables, it will display the variable on the screen UNLESS the assignment statement end with a semi-colon. It will also tell you the name of the variable, so the command

```
x=2+4
```

will display

```
x =
     6
```

on your screen, whereas the command

```
x=2+4;
```

displays nothing. If you ask MATLAB to make some computation but do not assign the result to a variable, MATLAB will assign it to an implicit variable called `ans` (short for "answer"). Thus the command

```
2+4
```

will display

```
ans =
     6
```

## B.2 Conditional Statements And Looping

As with any programming language MATLAB can evaluate boolean expression such as `A>B`, `A>=B`, `A<B`, `A<=B` and `A~=B` (the last one is not equal; `~` is MATLAB's negation operator). Also `~(A>B)`, `~(A<B)`, etc., can be used. These need to be used with a bit of case when $A$ and $B$ are not scalars, however. `A>B` creates a matrix of zeros and ones equal in size to $A$ and $B$. If you want to know is any of the elements of $A$ are bigger than any of the elements of $B$ is the same as checking whether any of the elements of the matrix `A>B` are non-zero.

MATLAB provides the functions any and all to evaluate matrices resulting form boolean expressions. As with many MATLAB functions, any and all operate on rows and return a rows vector with the same number of columns as the original matrix. This is true for sum and prod functions as well. The following are equivalent expressions

```
any(A>B)
```

and

```
sum(A>B)>0
```

The following are also equivalent:

```
all(A>B)
```

and

```
prod(A>B)>0
```

All of these expression are row vectors:

```
size(all(A>B))
```

is equal to

```
[1 max(size(A),size(B))]
```

Boolean expressions are mainly used to handle conditional execution of code using one of the following:

```
if expression
...
end

if expression
...
else
...
end
```

or

```
while expression
...
end
```

The first two of these are single conditionals, for example

```
if X>0, A=1/X; else A=0, end
```

You should also be aware of the switch command (type `help switch`). The last is for looping. Usually you use while for looping when you don't know how many times the loop is to be executed and use a for loop when you know how many times it will be executed. To loop through a procedure n times for example, one could use the following code:

```
for i=1:n, X(I)=3*X(i-1)+1; end
```

A common use of while for our purposes will be to iterate until some convergence criteria is met, such as

```
P=2.537;
X=0.5;
DX=0.5;
while DX<1E-7;
  DX=DX/2;
  if normcdf(X)>P, X=X-DX; else X=X+DX; end
  disp(X)
end
```

(can you figure out what this code does?). One thing in this code fragment that has not yet been explained is `disp(X)`. This will write the matrix $X$ to the screen.

## B.3 Scripts and Functions

When you work in MATLAB you are working in an interactive environment that stores the variables you have defined and allows you to manipulate them throughout a session. You do have the ability to save groups of commands in files that can be executed many times.

MATLAB has two kinds of command files, called m-files. The first is a script m-file. If you save a bunch of commands in a script file called `MYFILE.m` and then type the word `MYFILE` at the MATLAB command line, the commands in that file will be executed just as if you had run them each from the MATLAB command prompt (assuming MATLAB can find where you saved the file). A good way to work with

MATLAB is to use it interactively, and then edit you session and save the edited commands to a script file. You can save the session either by cutting and pasting or by turning on the diary feature (use the on-line help to see how this works by typing `help diary`).

The second type of M-files is the function file. One of the most important aspects of MATLAB is the ability to write your own functions, which can then be used and reused just like intrinsic MATLAB functions. A function file is a file with an m extension (e.g., `MYFUNC.m`) that begins with the word function.

```
function Z=DiagReplace(X,v)
% DiagReplace Put vector v onto diagonal of matrix X
% SYNTAX:
% Z=DiagReplace(X,v);
n=size(X,1);
Z=X;
ind=(1:n:n*n) + (0:n-1);
Z(ind)=v;
```

You can see how this function works by typing the following code at the MATLAB command line:

```
m=3; x=randn(m,m); v=rand(m,1); x, v, xv=diagreplace(x,v)
```

Any variables that are defined by the function that are not returned by the function are lost after the function has finished executing (`n` and `ind` in `DiagReplace`).

Here is another example:

```
function x = rndint(k,m,n)
% RANDINT Random integers between 1 and k (inclusive).
% SYNTAX:
%   x = rndint(k,m,n);
% Returns an m by n matrix
% Can be used for sampling with replacement.
x=ceil(k*rand(m,n));
```

Documentation of functions (and scripts) is very important. In M-files a `%` denotes that the rest of the line is a comment. Comments should be used liberally to help you and others who might read your code understand what the code is intending to do. The top lines of code in a function file are especially important. It is here where you should describe what the function does, what its syntax is and what each of the input and output variables are. These top line become an online help feature for your function. For example, typing `help randint` at the MATLAB command line would display the four commented lines on your screen.

A note of caution on naming files is in order. It is very easy to get unexpected results if you give the same name to different functions, or if you give a name that is already used by MATLAB. Prior to saving a function that you write, it is useful to use the `which` command to see if the name is already in use.

MATLAB is very flexible about the number of arguments that are passed to and from a function. This is especially useful if a function has a set of predefined defaults values that usually provide good results. For example, suppose you write a function that iterates until a convergence criteria is met or a maximum number of iterations has been reached. One way to write such a function is to make the convergence criteria and the maximum number of iterations be optional arguments. The following function attempts to find the value of $x$ such that $\ln(x) = ax$, where $a$ is a parameter.

```
function x=SolveIt(a,tol,MaxIters)
if nargin<3 | isempty(MaxIters), MaxIters=100; end
if nargin<2 | isempty(tol), tol=sqrt(eps); end
x=a;
for i=1:MaxIters
  lx=log(x);
  fx=x.*lx-a;
  x=x-fx./(lx+1);
  disp([x fx])
  if abs(fx)<tol, break; end
end
```

In this example, the command `nargin` means "number of input arguments" and the command `isempty` checks to see is a variable is passed but is empty (an empty variable is created by setting it to `[]`). An analogous function for the number of output arguments is `nargout`; many times it is useful to put a statement like

```
if nargout<2, return; end
```

into your function so that the function does not have do computations that are not requested. It is possible that you want nothing or more than one thing returned from a procedure. For example

```
function [m,v]=MeanVar(X)
% MeanVar Computes the mean and variance of a data matrix
% SYNTAX
%     [m,v]=MeanVar(X);
n=size(X,1);
m=mean(X);
```

```
if nargout>1
  temp=X-m(ones(n,1),:);
  v=sum(temp.*temp)/(n-1);
end
```

To use this procedure call it with `[mu,sig]=MeanVar(X)`. Notice that is only computes the variance if more than one output is desired. Thus, the statement `mu=MeanVar(X)` is correct and returns the mean without computing the variance.

In the following example, the function can accept one or two arguments and checks how many outputs are requested. The function computes the covariance of two or more variables. It can handle both a bivariate case when passed two data vectors and a multivariate case when passed a single data matrix (treating columns as variables and rows as observations). Furthermore it returns not only the covariance but, if requested, the correlation matrix as well.

```
function [CovMat,CorrMat]=COVARIANCE(X,Y)
% COVARIANCE Computes covariances and correlations
n=size(X,1);
if nargin==2
  X=[X Y]; % Concatenate X and Y
end
m=mean(X); % Compute the means
X=X-m(ones(n,1),:); % Subtract the means
CovMat=X'*X./n; % Compute the covariance
if nargout==2 % Compute the correlation, if requested
  s=sqrt(diag(CovMat));
  CorrMat=CovMat./(s*s');
end
```

This code executes in different ways depending on the number of input and output arguments used. If two matrices are passed in, they are concatenated before the covariance is computed, thereby allowing the frequently used bivariate case to be handled. The function also checks whether the caller has requested one or two outputs and only computes the correlation if 2 are requested. Although it would not be a mistake to just go ahead and compute the correlation, there is no point if it is not going to be used. Unless additional output arguments must be computed anyway, it is good practice to compute them only as needed. Some examples of calling this function are

```
c=COVARIANCE(randn(10,3));
[c1,c2]=COVARIANCE((1:10)',(2:2:20)');
```

Good documentation is very important but it is also useful to include some error checking in your functions. This makes it much easier to track down the nature of problems when they arise. For example, if some arguments are requires and/or their values must meet some specific criteria (they must be in a specified range or be integers) these things are easily checked. For example, consider the function `DiagReplace` listed above. This is intended for a square matrix $(n \times n)$ $X$ and an $n$-vector $v$. Both inputs are needed and they must be conformable. The following code puts in error checks.

```
function Z=DiagReplace(X,v)
% DiagReplace Put vector v onto diagonal of matrix X
% SYNTAX:
% Z=DiagReplace(X,v);
if nargin<2, error('2 inputs are required'); end
n=size(X,1);
if size(X,2)~=n, error('X is not square'); end
if prod(size(v))~=n, error('X and v are not conformable'); end
Z=X;
ind=(1:n:n*n) + (0:n-1);
Z(ind)=v;
```

The command error in a function file prints out a specified error message and returns the user to the MATLAB command line. An important feature of MATLAB is the ability to pass a function to another function. For example, suppose that you want to find the value that maximizes a particular function, say $f(x) = x * exp(-0.5x2)$. It would useful not to have to write the optimization code every time you need to solve a maximization problem. Instead, it would be better to have solver that handles optimization problems for arbitrary functions and to pass the specific function of interest to the solver. For example, suppose we save the following code as a MATLAB function file called `MYFUNC.m`

```
function fx=myfunc(x)
fx=x.*exp(-0.5*x.^2);
```

Furthermore suppose we have another function called `MAXIMIZE.m` which has the following calling syntax

```
function x=MAXIMIZE(f,x0)
```

The two arguments are the name of the function to be maximized and a starting value where the function will begin its search (this is the way many optimization routines work). One could then call `MAXIMIZE` using

```
x=maximize('myfunc',0)
```

and, if the MAXIMIZE function knows what it's doing, it will return the value 1. Notice that the word myfunc is enclosed in single quotes. It is the name of the function, passed as a string variable, that is passed in. The function MAXIMIZE can evaluate MYFUNC using the feval command. For example, the code

```
fx=feval(f,x)
```

is used to evaluate the function. It is important to understand that the first argument to feval is a string variable (you may also want to find out about the command eval, but this is only a primer, not a manual). It is often the case that functions have auxiliary parameters. For example, suppose we changed MYFUNC to

```
function fx=myfunc(x,mu,sig)
fx=x.*exp(-0.5*((x-mu)./sig).^2);
```

Now there are two auxiliary parameters that are needed and MAXIMIZE needs to be altered to handle this situation. MAXIMIZE cannot know how many auxiliary parameters are needed, however, so MATLAB provides a special way to handle just this situation. Have the calling sequence be

```
function x=MAXIMIZE(f,x0,varargin)
```

and, to evaluate the function, use

```
fx=feval(f,x,varargin{:})
```

The keyword varargin (variable number of input arguments) is a special way that MATLAB has designed to handle variable numbers of input arguments. Although it can be used in a variety of ways the simplest is shown here, where it simply passes all of the input arguments after the second on to feval. Don't worry too much if this is confusing at first. Until you start writing code to perform general functions like MAXIMIZE you will probably not need to use this feature in your own code, but it is handy to have an idea of what its for when you are trying to read other peoples' code.

## B.4 Debugging

Bugs in your code are inevitable. Learning how to debug code is very important and will save you lots of time and aggravation. Debugging proceeds in three steps. The first ensures that your code is syntactically correct. When you attempt to execute some code, MATLAB first scans the code and reports an error the first time it finds a

syntax error. These errors, known as compile errors, are generally quite easy to find and correct (once you know what the right syntax is).

The second step involves finding errors that are generated as the code is executed, known as run-time errors. MATLAB has a built-in editor/debugger and it is the key to efficient debugging of run-time errors. If your code fails due to run time errors, MATLAB reports the error and provides a trace of what was being done at the point where the error occurred. Often you will find that an error has occurred in a function that you didn't write but was called by a function that was called by a function that was called by a function (etc.) that you did write. A safe first assumption is that the problem lies in your code and you need to check what your code was doing that led to the eventual error.

The first thing to do with run-time errors is to make sure that you are using the right syntax in calling whatever function you are calling. This means making sure you understand what that syntax is. Most errors of this type occur because you pass the wrong number of arguments, the arguments you pass are not of the proper dimension or the arguments you pass have inappropriate values.

If the source of the problem is not obvious, it is often useful to use the debugger. To do this, click on File and the either Open or New from within MATLAB. Once in the editor, click on Debug, then on Stop if error. Now run your procedure again. When MATLAB encounters an error, it now enters a debugging mode that allows you to examine the values of the variables in the various functions that were executing at the tie the error occurs. These can be accessed by selecting a function in the stack on the editor's toolbar. Then placing your cursor over the name of a variable in the code will allow you to see what that variable contains. You can also return to the MATLAB command line and type commands. These are executed using the variables in the currently selected workspace (the one selected in the Stack). Generally a little investigation will reveal the source of the problem (as in all things, it becomes easier with practice). There is a third step in debugging.

Just because your code runs without generating an error message, it is not necessarily correct. You should check the code to make sure it is doing what you expect. One way to do this is to test it one a problem with a know solution or a solution that can be computed by an alternative method. After you have convinced yourself that it is doing what you want it to, check you documentation and try to think up how it might cause errors with other problems, put in error checks as appropriate and then check it one more time. Then check it one more time.

A few last words of advice on writing code and debugging.

1. Break your problem down into small chunks and debug each chunk separately. This usually means write lots of small function files (and document them).

2. Try to make functions work regardless of the size of the parameters. For exam-

ple, if you need to evaluate a polynomial function, write a function that accepts a vector of values and a coefficient vector. If you need such a function once it is likely you will need it again. Also if you change your problem by using a fifth order polynomial rather than a fourth order, you will not need to rewrite your evaluation function.

3. Try to avoid hard-coding parameter values and dimensions into your code. Suppose you have a problem that involves an interest rate of 7%. Don't put a lot of 0.07s into your code. Later on you will want to see what happens when the interest rate is 6% and you should be able to make this change in a single line with a nice comment attached to it, e.g.,

```
rho=0.07;                % the interest rate
```

4. Avoid loops if possible. Loops are slow in MATLAB. It is often possible to do the same thing that a loop does with a *vectorized* command. Learn the available commands and use them.

5. RTFM - internet lingo meaning Read The F-word of choice) Manual.

6. When you just can't figure it out, check the MATLAB technical support site (MathWorks), the MATLAB discussion group (comp.soft-sys.matlab) and DejaNews for posting about your problem and if that turns up nothing, post a question to the discussion group. Don't overdo it, however; people who abuse these groups are quickly spotted and will have their questions ignored. (If you are a student, don't ask the group to solve your homework problems. You will get far more out of attempting them yourself then you'll get out of having someone else tell you the answer. You are likely to be found out anyway and it is a form of cheating.)

## B.5 Other Data Types

So far we have only used variables that are scalars, vector or matrices. MATLAB also recognizes multidimensional arrays. Element by element arithmetic works as usual on these arrays (including addition and subtraction, as well as boolean arithmetic). Matrix arithmetic is not clearly defined for multidimensional arrays and MATLAB has not attempted to define a standard. If you try to multiply two multidimensional arrays, you will generate an error message. Working with multi-dimensional arrays can get a bit tricky but is often the best way to handle certain kinds of problems. An alternative to multi-dimensional arrays is what MATLAB calls a cell array. A

multidimensional array contains numbers for elements. A cell array is an array (possibly a multi-dimensional one) that other data structures as elements. For example, you can define a $2 \times 1$ cell array that contains a $3 \times 1$ matrix in it first cell (i.e., as element $(1,1)$) and a $4 \times 4$ matrix in its second cell. Cell arrays are defined using curly brackets rather than square ones, e.g.,

```
x={[1;2],[1 2;3 4]};
```

Other data types are available in MATLAB include string variables, structure variables and objects. A string variable is self-explanatory. Structure variables are variables that have named fields that can be referenced. For example, a structure variable, `X`, could have the fields `DATE` and `PRICE`. One could then refer to the data contained in these filed using `X.DATE` and `X.PRICE`. If the structure variable is itself an array, one could refer to fields of an element in the structure using `X(1).DATE` and `X(1).PRICE`. Object type variables are like structures but have methods attached to them. The fields of an object cannot be directly accessed but must be access using the methods associated with the object. Structures and objects are advanced topics that are not needed to get started using MATLAB. They are quite useful if you are trying to design user friendly functions for other users. It is also useful to understand objects when working with MATLAB's graphical capabilities, although, again, you can get pretty nice plots without delving into how objects work.

## B.6 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 your code to do what you want 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 derivatives of a function numerically. Such a function will be used extensively in this book.

# Web Resources

http:// Website for this text
http:// Website for CompEcon toolbox

# References

Ames, William F. *Numerical Method for Partial Differential Equations*, 3rd ed. San Diego, CA: Academic Press, 1992.

Antonsev, S.N., K.-H. Hoffman, and A.M. Khludnev, editors. *Free Boundary Problems in Continuum Mechanics*, vol. 106 of *International Series of Numerical Mathematics*. Basel: Birkhäuser Verlag, July 15–19, 1992.

Atkinson, K.E. *An Introduction to Numerical Analysis*, 2nd ed. New York: John Wiley and Sons, 1989.

Balinski, M.L. and R.W. Cottle, editors. *Complementarity and Fixed Point Problems*, no. 7 in Mathematical Programming Studies. Amsterdam: North-Holland, 1978.

Bharucha-Reid, A.T. *Elements of the Theory of Markov Processes and Their Applications*. New York: McGraw-Hill, 1960.

Black, Fischer and Myron Scholes. "The Pricing of Options and Corporate Liabilities." *Journal of Political Economy* 81(1973):637–654.

Brennan, Michael J. and Eduardo S. Schwartz. "Evaluating Natural Resource Investments." *Journal of Business* 58(1985):135–157.

Chiang, Alpha C. *Elements of Dynamic Optimization*. Prospect Heights, IL: Waveland Press, 1999.

Cottle, R.W., F. Giannessi, and J-L. Lions, editors. *Variational Inequalities and Complementarity Problems*. Chichester: John Wiley and Sons, 1980.

Cottle, R.W., J.-S. Pang, and R.E. Stone. *The Linear Complementarity Problem*. San Diego: Academic Press, 1992.

Cox, D.R. and H.D. Miller. *The Theory of Stochastic Processes*. New York: John Wiley and Sons, 1965.

Cox, John C., Jonathan E. Ingersoll, and Stephen A. Ross. "A Theory of the Term Structure of Interest Rates." *Econometrica* 53(1985):385–407.

Dai, Qiang and Kenneth J. Singleton. "Specification Analysis for Affine Term Structure Models." *Journal of Finance* 55(2000):forthcoming.

Dennis, Jr., J.E. and R.B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Englewood Cliffs, NJ: Prentice-Hall, 1983.

Dixit, Avinash K. and Robert S. Pindyck. *Investment Under Uncertainty*. Princeton, NJ: Princeton University Press, 1994.

Dixit, Avinash. "A Simplified Treatment of the Theory of Optimal Regulation of Brownian Motion." *Journal of Economic Dynamics and Control* 15(1991):657–673.

Dixit, Avinash. *The Art of Smooth Pasting*, vol. 55 of *Fundementals of Pure and Applied Economics*. Chur, Switzerland: Harwood Academic Publishers, 1993a.

Dixit, Avinash. "Choosing Among Alternative Discrete Investment Projects Under Uncertainty." *Economics Letters* 41(1993):265–268.

Dorfman, R. "An Economic Interpretation of Optimal Control Theory." *American Economic Review* 59(1969):817–831.

Duffie, Darrell and Rui Kan. "A Yield-Factor Model of Interest Rates." *Mathematical Finance* 6(1996):379–406.

Duffie, Darrell. *Dynamic Asset Pricing Theory*, 2nd ed. Princeton, NJ: Princeton University Press, 1996.

Dumas, Bernard. "Super Contact and Related Optimality Conditions." *Journal of Economic Dynamics and Control* 15(1991):675–685.

Fackler, Paul L. "Multivariate Option Pricing." North Carolina State University, 2000.

Feller, William. "Diffusion Problems in Genetics," *Proceedings of the Second Symposium on Mathematical Statistics and Probability*, pp.227–246. Berkeley: July/August, 1950.

Feller, William. "Two Singular Diffusion Problems." *Annals Of Mathematics* 54(1)(1951):173–182.

Ferris, Michael C. and Todd S. Munson. "Interfaces to PATH3.0: Design, Implementation and Usage." *Computational Optimization and Applications* 12(1999):207–227.

Ferris, M.C. and J.S. Pang. "Engineering and Economic Applications of Complementarity Problems." *SIAM Review* 39(1997):669–713.

Ferris, Michael C. and Krung Sinapiromsaran "Formulating and Solving Nonlinear Programs as Mixed Complementarity Problems," *Optimization*, vol. 481 of *Lecture Notes in Economics and Mathematical Systems*, ed. V.H. Nguyen, J.J. Strodiot, and P. Tossings, . New York: Springer-Verlag, 2000.

Ferris, Michael C., Michael Mesnier, and Jorge J. More. "The NEOS Server for Complementarily Problems: PATH." Computer Sciences Department, University of Wisconsin, Madison, WI, June, 1996.

Ferris, Michael C.and Christian Kanzow. "Complementarity and Related Problems: A Survey." University of Wisconsin - Madison, 1998.

Fleming, Wendell H. and Raymond W. Rishel. *Deterministic and Stochastic Optimal Control*, no. 1 in Applications of Mathematics. New York: Springer-Verlag, 1975.

Fletcher, R. *Practical Methods of Optimization*, 2nd ed. New York: John Wiley and Sons, 2000.

Gaffney, M. "Concepts of Financial Maturity of Timber and Other Assets.". Department of Agricultural Economics, North Carolina State CollegeA.E. Information Series 62, 1960.

Gill, P.E., W. Murray, and M.H. Wright. *Practical Optimization*. NewYork: Academic Press, 1981.

Goldman, M. Barry, Howard B. Sosin, and Mary Ann Gatto. "Path Dependent Options: "Buy at the Low, Sell at the High"." *Journal of Finance* 34(1979):1111–1127.

Golub, Gene H. and James M. Ortega. *Scientific Computing and Differential Equations: An Introduction to Numerical Methods*. San Diego: Academic Press, 1992.

Golub, G.H. and C.F. van Loan. *Matrix Calculations*, 2nd ed. Madison, WI: University of Wisconsin Press, 1989.

Hershleifer, J. *Investment, Interest and Capital*. Englewood Cliffs, NJ: Prentice Hall, 1970.

Hoffman, K.H. and J. Sprekels, editors. *Free Boundary Problems: Theory and Applications II*, no. 186 in Pitman Research Notes in Mathematics. Essex, England: Longman Scientific and Technical, 1990.

Hull, John C. *Options, Futures and Other Derivative Securities*, 4th ed. Englewood Cliffs, NJ: Prentice Hall, 2000.

Judd, Kenneth L. *Numerical Methods in Economics*. Cambridge, MA: MIT Press, 1998.

Kamien, M.I. and N.L. Schwartz. *Dynamic Optimization: The Calculus of Variations and Optimal Control in Economics and Management*, 2nd ed. New York: North-Holland, 1981.

Karlin, Samuel and Howard M. Taylor. *A Second Course in Stochastic Processes*, 2nd ed. New York: Academic Press, 1981.

Kennedy, William J. and James E. Gentle. *Statistical Computing*, vol. 33 of *Statistics: Textbooks and Monographs*. New York: Macel Dekker, 1980.

Kremers, Hans and Dolf Talman. "A New Pivoting Algorithm for the Linear Complementarity Problem Allowing for an Arbitrary Starting Point." *Mathematical Programming* 63(1994):235–252.

Kushner, H.J. and P.G. Dupuis. *Numerical Methods for Stochastic Control Problems in Continuous Time*. New York: Springer-Verlag, 1992.

Leon, Steven J. *Linear Algebra with Applications*. New York: Macmillan Publishing Co., 1980.

Ludwig, Donald and James M. Varrah. "Optimal Harvesting of a Randomly Fluctuating Resource. II. Numerical Methods and Results." *SIAM Journal of Applied Mathematics* 37(1)(1979):185–205.

Ludwig, Donald. "Optimal Harvesting of a Randomly Fluctuating Resource. I. Application of Perturbation Methods." *SIAM Journal of Applied Mathematics* 37(1)(1979):166–184.

Lund, D. and B. Oksendal, editors. *Stochastic Models and Option Values*. New York: North Holland, 1991.

Majd, Saman and Robert S. Pindyck. "Time to Build, Option Value, and Investment Decisions." *Journal of Financial Economics* 18(1987):7–28.

Majd, Saman and Robert S. Pindyck. "The Learning Curve and Optimal Production Under Uncertainty." *Rand Journal of Economics* 20(3)(1989):331–343.

Malliaris, A.G. and W.A. Brock. *Stochastic Methods in Economics and Finance*, vol. 17 of *Advanced Textbooks in Economics*. Amsterdam: North-Holland, 1982.

Mangel, Marc. *Decision and Control in Uncertain Resource Systems*. Orlando, FL: Academic Press, 1985.

McDonald, Robert L. and Daniel R. Siegel. "Investment and the Valuation of Firms When There is an Option to Shut Down." *International Economic Review* 26(2)(1985):331–349.

Merton, Robert C. "Lifetime Portfolio Selection Under Uncertainty: The Continuous-Time Case." *Review of Economics and Statistics* 51(1969):247–257.

Merton, Robert C. "Optimum Consumption and Portfolio Rules in a Continuous-Time Model." *Journal of Economic Theory* 3(1971):373–413.

Merton, Robert C. "The Theory of Rational Option Pricing." *Bell Journal of Economics and Management Science* 4(1973):141–183.

Merton, Robert C. "An Asymptotic Theory of Growth Under Uncertainty." *Review of Economic Studies* 42(1975):375–393.

Neftci, Salih N. *An Introduction to the Mathematics of Financial Derivatives.* San Diego, CA: Academic Press, 1996.

Ortega, J.M. and W.C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables.* New York: Academic Press, 1970.

Pindyck, Robert S. "Uncertainty in the Theory of Renewable Resource Markets." *Review of Economic Studies* 51(1984):289–303.

Press, William H., Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes*, 2nd ed. Cambridge: Cambridge University Press, 1992.

Shimko, David C. *Finance in Continuous Time: A Primer.* Florida: Kolb Publishing Company, 1992.

Smith, Vernon L. "On Models of Commercial Fishing." *Journal of Political Economy* 77(3)(1969):181–198.

Wilmott, Paul. *Derivatives: The Theory and Practice of Financial Engineering.* Chichester, England: John Wiley and Sons, 1998.

# Index

affine asset pricing model, 337
algorithms
    bisection, 31
    Broyden's, 40
    Euler's method, 115
    function iteration, 33
    golden search, 64
    line search, 74
    method of scoring, 78
    Nelder-Mead, 66
    Newton's, 35
    Newton-Raphson, 68
    quasi-Newton, 70
    Runge-Kutta methods, 115
    secant method, 39
Armijo search, 74

bisection method, 31
Broyden's method, 40–43

Cholesky factorization, 22
complementarity problems, 48–55
Cournot oligopoly model, 37

derivatives, numerical, 107
diagonally dominant matrix, 24

Euler's method, 115

finite difference methods, 107
fixed-point problems, 33

Gauss-Hermite quadrature, 99
Gauss-Jacobi, 24
Gauss-Legendre quadrature, 97
Gauss-Seidel, 24
Gaussian elimination, 17
Gaussian quadrature, 97
golden search method, 64
Goldstein search, 75

initial value problems, 114

low discrepency sequences, 102

maximum likelihood estimation, 78
method of scoring, 78
Monte Carlo Integration, 100
multivariate methods
    function approximation, 145–149
    quadrature, 96, 99, 106
multivariate models, 337

Nelder-Mead algorithm, 66
Newton's method, 35–37
Newton-Raphson method, 68
nonlinear least squares, 77
numerical calculation of expectations, 98, 106
    beta distribution, 107
    gamma distribution, 107
    lognormal distribution, 100, 107
    Normal distribution, 99, 106

option pricing models
    exotic options, 334
options
    Asian, 334

barrier, 336
down-and-out, 336
lookback, 336
ordinary differential equations, 114

pseudo random sequences, 100

quadrature
Guassian, 97
Monte Carlo, 100
Newton-Cotes, 95
quasi-Monte Carlo, 102
Simpson's rule, 96
trapezoid rule, 95
Quasi-Monte Carlo Integration, 102
quasi-Newton methods, 39–45, 70
secant method, 39

random number generators, 100
renewable resource models, 117
Runge-Kutta methods, 115

secant method, 39–40

tensor products, 96, 99, 106, 145–149