6/12/2005
06:34

Computational Economics

David A. Kendrick

P. Ruben Mercado

Hans M. Amman

Contents

Preface

One of the best ways to learn computational economics is to do computational economics. One of the best ways to do computational economics is to begin with existing models and modify them as you experiment with them. This is the approach used in this book.

In each chapter an economic model is presented. First the economics and mathematics of the model is discussed and then the computational form of the model is analyzed. This process enables one to learn the economics and the mathematics of the problem area as well as the computational methods that are used in that area. For example, in the economic growth area we make use of a Ramsey type model. The economics of growth theory are first discussed along with the equations that model this process. Then the software representation of the model is presented so that the reader can see how the model can be solved on a computer. The student can then modify the model in order to analyze its sensitivity to various parameters and functional specifications. In the process of experimenting with the model one can gain an improved understanding of both the software and of the economic modeling.

This book grew out of undergraduate and graduate level courses on computational economics taught by us at the University of Texas, ISEG (Argentina) and the University of Amsterdam. Also, a number of teaching assistants and undergraduate students participated in the development of chapters, notably Daniel Gaynor and Genevieve Solomon.

This book is intended for use by advanced undergraduates and professional economists and even, as a first exposure to Computational Economics, for graduate students. We expect the development in coming years of undergraduate courses with a focus on economic modeling along the lines outlined in this book. Also, we envisage the development of a two course sequence in Computational Economics in graduate programs. The introductory course would have a broad economic modeling focus with an approach similar to that used in some chapters of this book. The second course would focus on algorithms and numerical methods.

Part of our motivation for writing this book is spelled out in a couple of paragraphs that are taken from a paper the three of us wrote with the title “Computational

Economics: Help for the Underestimated Undergraduate". [1] These comments – though written for that paper – apply equally as well to this book.

> "The ubiquitous personal computer has filtered deeply through the lives of college undergraduates; however undergraduate education in economics has so far failed to take full advantage of this sweeping change. We are underestimating the learning ability and insufficiently challenging a whole generation of undergraduate students in economics. Our thesis is that computational economics offers a way to improve this situation and to bring new life into the teaching of economics in colleges and universities.
>
> With its early focus on algorithms, computational economics seemed well-suited for a relatively small group of graduate students and unlikely to have much impact on undergraduates. However, that is changing as we are discovering that computational economics provides an opportunity for some students to move away from too much use of the lecture-exam paradigm and more use of a laboratory-paper paradigm in teaching undergraduate economics. This opens the door for more *creative* activity on the part of the students by giving them models developed by previous generations and challenging them to modify those models. The modifications can be altering the models to make them applicable to the student's interest or finding weaknesses in the model that can be strengthened by changes in the structure of the model. In the process the students become much more involved in their own education."

The organization of the chapters in the book reflects primarily the outline of the courses at the University of Texas. The aim is to let the students find an area of computational economics that interest them and to pursue that area. Since some of the students are interested in microeconomics, others in macroeconomics and others in finance an effort is made to give a quick and broad exposure to models across a range of fields early in the semester. Then the range is covered again later in the semester in greater depth. The book is structured to follow this pattern. In Part I there is a "once over lightly" treatment of computational economics examples from a number of fields. This is then repeated in greater depth and complexity in Part II. Part III covers an

[1] Kendrick, Mercado and Amman (2005).

advanced area that is of special interest to the authors, namely the solution of macroeconomic models with stochastic control methods.

We would like to thank Alan Manne, Manfred Gilli and other reviewers for comments on earlier drafts of this book that helped us to substantially improve it. Also, we want to thank Provost Sheldon Ekland-Olson and Dean Brian Roberts of the University of Texas for funding which was used to support preparation of some of the materials in this book. In addition, we would like to thank Peter Dougherty of the Princeton University Press for his encouragement of the development of this book over a period of many years.

Thanks are due to a number of undergraduate and graduate students who took the computational economics courses at the University of Texas and contributed ideas and models which added to the quality of several of the chapters and who helped to create and maintain the web sites, viz. Pichit Akrathit, Joe Breedlove, Michael Evanchik, Shyam Gouri-Suresh, Miwa Hattori, Carter Hemphill, Kyle Hood, Seung-Rae Kim, Kevin Kline, Paul Maksymonko, Juan Carlos Navarro and Huber Salas.

Introduction

One can think of learning computational economics by following one of three different routes - via computational methods, via mathematical methods or via economic areas. The computational methods route would focus on the use of a particular computer software system like MATLAB or Mathematica and teach the students the capabilities of those languages with examples from economics. The mathematical route would focus on algorithms to solve various classes of mathematical models such as linear or nonlinear programming models, differential or difference equations, and dynamic programming models and provide examples of the use of each kind of model in economics. The economic areas approach would focus on microeconomics, macroeconomics, finance, game theory, environmental economics etc. and teach the students how to formulate and solve economic models in each of these areas. For this book we have chosen the last of these three approaches.

Thus this is a book about *computational economics*, but also about *economic modeling*. As a student approaches a new area of interest we want to help him or her first think through the economics of the subject. Then we develop this economics into a mathematical model. Finally we specify the mathematical model as a computational model in a particular software system. We believe that this process can be greatly facilitated by encouraging the students to follow Professor Paul Samuelson's advice and "stand on the shoulders" of those who have gone before. This is done by beginning from subject areas and problems that other economists have studied and learning how the economics was converted to mathematics and then to computational models in those areas.

Therefore this book is organized around economic topics rather than around mathematical or computational topics. However, we did not put all the microeconomics in the first section, then the macroeconomics etc. Rather the book is divided into two rounds of relatively simple models and then more complex models as was discussed above in the Preface.

Software Systems

Students who begin studying computational economics frequently ask the question, "What programming language should I learn?" [2] The answer given in this book is to first become *acquainted* with a number of high-level languages such as GAMS, Mathematica, MATLAB and Duali as well as the Solver in Excel and the Access database software. Moreover, it is useful to become acquainted with each of these software systems in the midst of solving the kind of economic models that are naturally developed in each of these systems. Then later one can dig deeper into one or more of the software systems and gain some level of mastery of it while writing a short mid-term paper, a term paper or doing research. At a still later stage, students who find that they have a continuing interest in computational economics would be well advised to progress to lower level languages such as Visual Basic, Fortran, C, C++, C# or Java.

There are different types of software paradigms, each of them more or less suitable to represent specific types of models. In this book, we present a selected set of high-level software systems, each corresponding to a specific paradigm.

We start the book with relatively simple models represented in Excel ("spreadsheet paradigm") as a way of beginning with a software paradigm that is well known and accessible to almost everybody, since this software system is available on most PC's. Excel is useful to solve small models that do not involve simultaneous systems of equations; however, is not well designed for vector-matrix operations. For this type of operations we will use MATLAB later in the book. However, Excel has a nonlinear optimization solver which can handle constrained optimization problems and is very handy to set up and solve interesting models such as a Ramsey type model of economic growth and a small neural net.

Also, early in the book we introduce Access ("relational database paradigm"), which like Excel is a very accessible software system. Access is well suited to develop relatively simple relational databases and its use is illustrated with a prototype U.S. database.

The "set driven" paradigm is introduced with GAMS. This software system, particularly well suited to deal with medium and large size models involving from tens to hundreds of variables and equations, allows us to specify problems in an organized and compact way, defining sets to be used as indexes, and specifying scalars, parameters, variables and equations in a parsimonious way. We solve with GAMS models of

[2] For a discussion of some of the software systems used in economics see Amman and Kendrick (1999b).

transportation, financial planning, general equilibrium, macroeconomics and global warming.

The "vector matrix" paradigm is introduced with MATLAB. This software system is useful to deal with models or problems involving intensive use of vector and matrix operations, cell arrays and data structures, and also to deal with problems of recursive structure requiring intensive use of "loops". We use MATLAB to solve problems of portfolio optimization, genetic algorithms, agent-based models and dynamic programming.

The "symbolic math" paradigm is introduced with Mathematica. This software system is particularly powerful to solve symbolic algebra and calculus problems, and we use it to represent partial equilibrium and game theoretic problems.

Finally, in a Special Topics Section in Part III of the book, and by means of macroeconomic applications we introduce Duali, a "dialog box driven" software designed to solve stochastic control and dynamic policy analysis problems. The basic code of this software is written in C, and contains a variety of simple and complex quadratic linear dynamic programming algorithms.

Most economics departments and economics students already have many software systems available on their computers and hopefully will also have the ability to acquire most of the rest of those used in this book. We have provided in our web site at

http://www.eco.utexas.edu/compeco

the input files for the economic models that are used in this book. Also, this web site contains pointers to software sources, supporting books and user guides. In an effort to keep student cost down, we have endeavored to keep most of the models used in this book small enough that they can be solved with the student versions of the software systems.

With the exception of Duali, all of the software systems we use are commercial products. In contrast, the Duali software is academic software which is under development by two of us (Kendrick and Amman) and has no support staff or help desk. It is designed to greatly reduce the learning curve for developing dynamic deterministic and stochastic optimization models and is a most useful starting point into economic research in these areas. However, it is early in its stages of development and must be used with caution.

Numerical Methods

In this book we present not only a variety of models and software paradigms, but also introductions to diverse numerical methods needed to solve them. As with the software systems, we think that is useful to become acquainted with each of those numerical methods in the midst of solving the kind of economic models that are naturally involved with each of these methods.

A number of the models presented in the book are solved with linear programming methods or nonlinear optimization methods based on gradient and/or Newton methods. Thus we provide an introduction to these methods in appendices at the end of the book. Other methods are introduced directly in particular chapters. Neural nets are applied to a stock price prediction problem, Monte Carlo methods are applied to a portfolio selection problem and genetic algorithms are applied to an evolutionary game and to a portfolio selection problem. Quadratic linear dynamic programming is illustrated with a simple macroeconomic policy analysis application. Finally, the Fair and Taylor iterative method to solve rational expectations models, together with the Amman and Kendrick method to solve optimal control models with forward looking variables is applied to a prototype macro model developed by Taylor.

Teaching Methods

A description of the teaching methods used in the computational economics courses at the University of Texas will help the reader to understand the way in which the materials in this book have been developed. One aspect of these courses is that they have a weekly cycle. As was described above, the first class each week is on the economic theory and mathematical model of the subject for the week. The second class is on the computational methods used to solve the model. The third class of the week is not in a lecture room but rather in a computer laboratory where the students are ask to solve the base model and then to modify (and solve) the model several times in order to study its structure and operation. One week after the computer laboratory class the students are asked to turn in a short paper a few pages in length that describes their own experiments with the model during the week and the results obtained. The weekly teaching cycle is reflected in this book with some suggested experiments listed at the end of each chapter. However, the students are encouraged to strike out on their own – a process which enhances both enjoyment and learning.

Since the emphasis in these computational economics courses is on creativity, there is both a mid-term paper and a final paper. The students are asked in the mid-term paper to modify one of the models from the course or to select an existing model from the GAMS library or another similar source and then to make minor improvements in the model. In the final paper they are asked to carry this process forward and make major modifications to an existing model or to create a model of their own.

Several alternative approaches to the one used in this book are available for the study of computational economics. However, until now most books in this field have focused on graduate level instruction while we are hoping to be helpful to both undergraduates and graduate students. For an approach using the GAMS software exclusively and focusing on linear and nonlinear programming methods see Thompson and Thore (1992). For approaches using numerical methods see Judd (1998) who uses several computer languages or Miranda and Fackler (2002) who use MATLAB. Varian (1993a) and (1996) presents a variety of models in Mathematica. For a web site that supports a course on applied macroeconomics using computational methods taught by Prof. Harris Dellas at the University of Bern that is somewhat similar to the approach taken in this book see

http://www.vwi.unibe.ch/amakro/Lectures/computer/

For books that focus on numerical methods in macroeconomics with some applications in MATLAB see Marimon and Scott (1999) and Adda and Cooper (2003). For a book with a collection of articles that consider a variety of numerical methods to solve macroeconomic models see Hughes Hallett and McAdam (1999). For a handbook with a collection of articles about computational economics see Amman, Kendrick and Rust (1996). Also, you are encouraged to browse the Internet site of the Society for Computational Economics at

http://comp-econ.org

where you will find information about meetings, journals and book series.

Given the array of materials that are becoming available for teaching computational economics, we are hopeful that courses in this field will become a part of the core curriculum in both undergraduate and graduate education in economics as happened before with Mathematical Economics and Econometrics. Moreover, we hope our book will motivate and help instructors in those areas to offer courses in Computational Economics. We are aware of courses in Computational Economics that have been offered in recent years at Stanford, Yale, Maryland, Ohio State, Bern, Harvard and Texas and believe that some of these courses will migrate toward the core as courses are added at other universities.

Part I

Once Over Lightly . . .

Chapter 1

Growth Model in Excel

Most economists are familiar with the spreadsheet and even with the database capabilities of the Excel software, but fewer are aware that Excel also contains powerful solution procedures for solving both linear and nonlinear programming problems. Because the Excel interface is so familiar to many and because the specification of programming problems in Excel is relatively straightforward, there are times when Excel is the software of choice for solving certain types of optimization problems. In particular, when the models are small enough that the set driven nature of GAMS does not give it an advantage over Excel, it may be advantageous to solve optimization problems in Excel. To illustrate this we will use a one-sector growth model of the type that is widely used in the economics literature.

The model we will use is the famous Ramsey model of economic growth. Models of this type have been widely used in the economic growth literature. In particular, we will follow the versions developed by Chakravarty (1962) and Taylor and Uhlig (1990). We will employ a finite horizon version with a terminal capital stock constraint.

The model will first be introduced in a mathematical form and then in a computational form.[3] The essential economics of the simple growth model used in this chapter is a trade-off between consumption and investment. More consumption in a time period means more utility in that time period but less investment and therefore less capital stock and less production in future time periods. So the key elements of the model are the production function with capital being used to produce output, the capital accumulation relationship with investment creating new capital and the utility function with consumption resulting in utility.

---

[3] Most models used in this book cannot be solved analytically so numerical methods are required. However, even when analytical solutions can be obtained, as shown later in Appendix J, it is still useful to obtain numerical solutions so that the code can be checked on simple models. Then the numerical methods can be used with more confidence when they are applied to more complex models that cannot be solved analytically.

1 Mathematical Form

The production side of the economy is specified in a stylized form by means of an aggregate production function

(1) $$Y_t = \theta \ K_t^{\alpha}$$

where

Y_t = output in period t

θ = a technology parameter

K_t = the capital stock in period t

α = exponent of capital in the production function

This is the widely used Cobb-Douglas form of a production function except that function usually includes both capital and labor inputs. However, for the sake of simplicity, the production function in this model includes only capital.

Consider next the capital accumulation constraint

(2) $$K_{t+1} = K_t + Y_t - C_t$$

where

C_t = consumption in period t

which says that the capital stock next period will be the same as this period plus the difference between output and consumption which is saving or investment. For the moment depreciation of the capital stock is ignored though you might want to add that to the model in an experiment.

Also, the production function (1) can be substituted into the capital accumulation equation (2) to obtain the equation

(3) $$K_{t+1} = K_t + \theta \ K_t^{\alpha} - C_t$$

In addition, the model has an initial condition that specifies the size of the capital stock in the initial period.

(4) K_0 given.

The model also includes a terminal condition that fixes a minimum amount of capital that must be left to the next generation after the time horizon covered by the model.

(5) $$K_N \geq K^*$$

where

K^* = a lower bound on the amount of capital required in the terminal period, N.

Finally, the model has a criterion function that is the discounted value of the utility that is obtained from consumption over all of the periods covered by the model. It is written in two steps. First the utility in each period is defined as

(6) $$U(C_t) = \frac{1}{(1-\tau)} C_t^{(1-\tau)}$$

where

$U(C_t)$ = the utility in period t as a function of consumption in that period

τ = a parameter in the utility function[4]

Then the sum of the discounted utilities is specified as

(7) $$J = \sum_{t=0}^{N-1} \beta^t \; U(C_t)$$

where

J = the criterion value

β = the discount factor = $\frac{1}{1+\rho}$

ρ = the discount rate

and the substitution of Eq. (6) into Eq. (7) yields the criterion function

[4] This is a popular form of the utility function which is known as the "constant elasticity of intertemporal substitution" function. Roughly speaking, think of the elasticity of intertemporal substitution as measuring the degree of substitutability between consumption "today" and "tomorrow" or, in geometric terms, measuring the curvature of the indifference curves corresponding to consumption at any two points in time. For this function, the elasticity of substitution is constant and equal to $1/\tau$.

(8) $$J = \sum_{t=0}^{N-1} \beta^t \frac{1}{(1-\tau)} C_t^{(1-\tau)}$$

In summary, the model consists of the criterion function (8), the capital accumulation equation (3) and the initial and terminal conditions (4) and (5) and can be stated as find

$(C_0, C_1, \cdots, C_{N-1})$ to maximize

(8) $$J = \sum_{t=0}^{N-1} \beta^t \frac{1}{(1-\tau)} C_t^{(1-\tau)}$$

subject to

(3) $$K_{t+1} = K_t + \theta \ K_t^{\alpha} - C_t$$

(4) K_0 given.

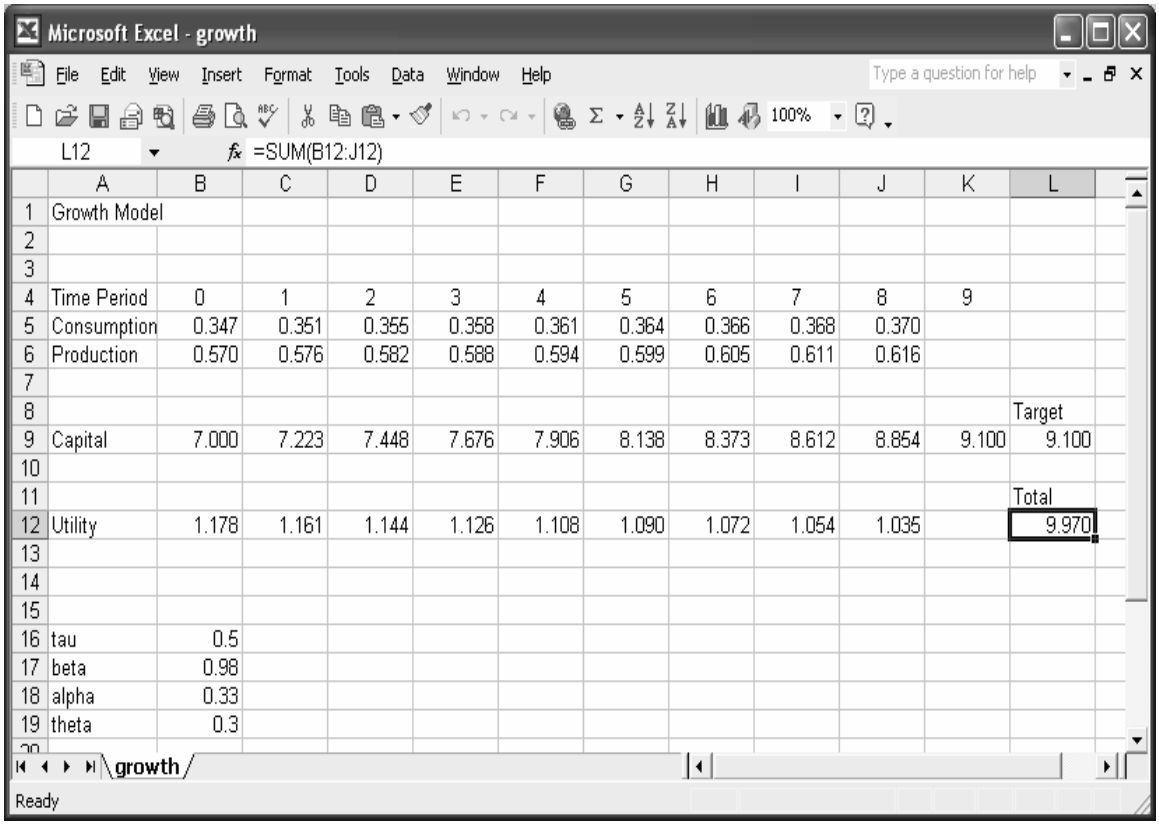
(5) $K_N \geq K^*$

So the essential problem is to choose those levels of consumption, over the time periods covered by the model that strike the right balance between consumption and investment. Lower consumption in any given period means less utility in that period but more savings and therefore larger capital stocks and more production in future years.

This growth model is a nonlinear programming problem because of the nonlinearities in the criterion function (8) and the capital accumulation equation (3). It can be stated and solved rather nicely in Excel as is discussed below.

2. Computational Form

Consider first a spreadsheet layout of the model as shown in Figure 1.1 below. The corresponding Excel file is in the book web page.



	A	B	C	D	E	F	G	H	I	J	K	L
1	Growth Model											
2												
3												
4	Time Period	0	1	2	3	4	5	6	7	8	9	
5	Consumption	0.347	0.351	0.355	0.358	0.361	0.364	0.366	0.368	0.370		
6	Production	0.570	0.576	0.582	0.588	0.594	0.599	0.605	0.611	0.616		
7												
8												Target
9	Capital	7.000	7.223	7.448	7.676	7.906	8.138	8.373	8.612	8.854	9.100	9.100
10												
11												Total
12	Utility	1.178	1.161	1.144	1.126	1.108	1.090	1.072	1.054	1.035		9.970
13												
14												
15												
16	tau	0.5										
17	beta	0.98										
18	alpha	0.33										
19	theta	0.3										

Figure 1.1 Growth Model in Excel with Total Utility Highlighted

Notice first that the model horizon covers time periods numbered from zero through nine so that period zero will be the initial period and period nine will be the terminal period. The rows below the time periods displays the

consumption, C_t

production, Y_t

capital stock, K_t

utility, $U(C_t)$

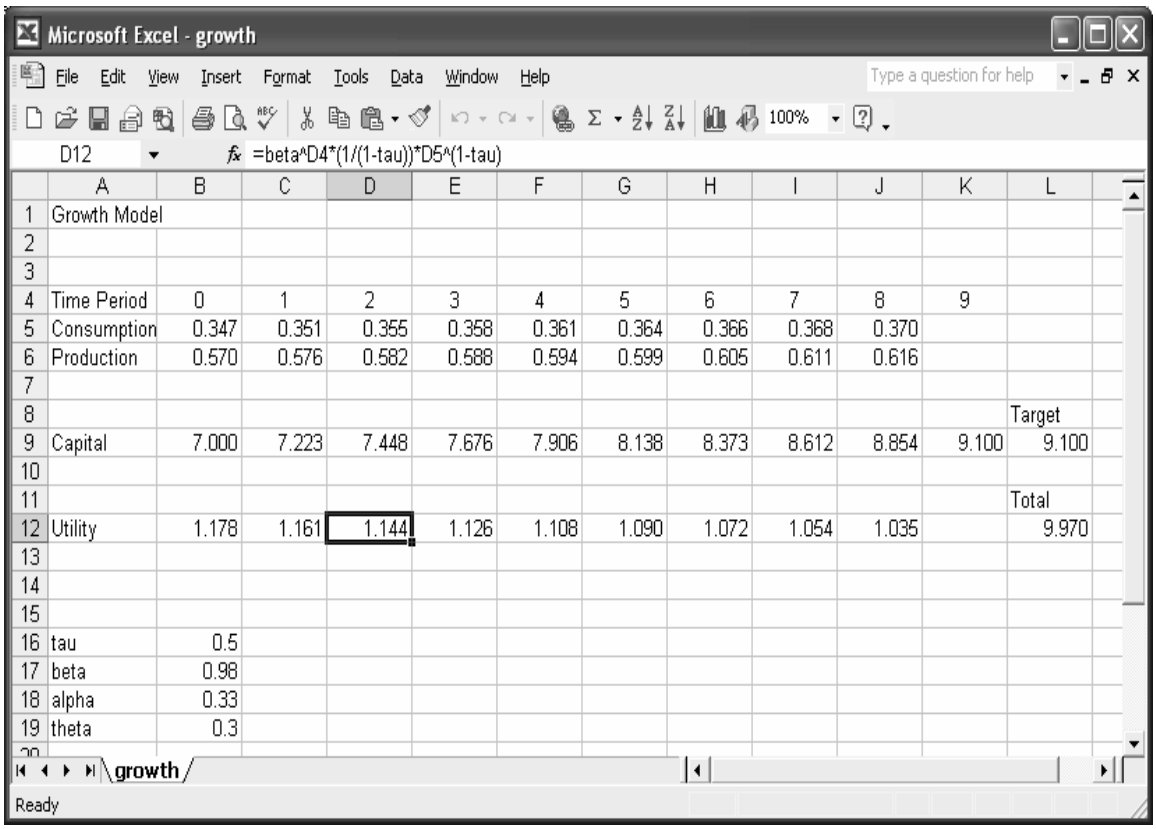
in each time period. All of these values are calculated when the model is solved and we shall show shortly how the calculations are structured. However, for now look only at

the cell below the "Total" label, i.e. cell L12, which is highlighted in the bottom right corner of the spreadsheet. It contains the value 9.97; however, we are not so much interested in that value as in how it is obtained. Look at the formula bar at the top of the spreadsheet which contains the expression

```
SUM (B12:J12)
```

This indicates that this cell contains the sum of the utility values for periods zero through eight which are contained in the cells B12 through J12.

Actually, the value in each of the cells B12 through J12 is not, strictly speaking, the utility for each period but rather the *discounted* utility for each period. This is illustrated in Figure 1.2 below.



	A	B	C	D	E	F	G	H	I	J	K	L
1	Growth Model											
2												
3												
4	Time Period	0	1	2	3	4	5	6	7	8	9	
5	Consumption	0.347	0.351	0.355	0.358	0.361	0.364	0.366	0.368	0.370		
6	Production	0.570	0.576	0.582	0.588	0.594	0.599	0.605	0.611	0.616		
7												
8												Target
9	Capital	7.000	7.223	7.448	7.676	7.906	8.138	8.373	8.612	8.854	9.100	9.100
10												
11												Total
12	Utility	1.178	1.161	1.144	1.126	1.108	1.090	1.072	1.054	1.035		9.970
13												
14												
15												
16	tau	0.5										
17	beta	0.98										
18	alpha	0.33										
19	theta	0.3										

Figure 1.2 The Calculation of Discounted Utility in Each Period

The cell D12 in the utility row is highlighted and the expression which is used to calculate the value in that cell is displayed in the formula bar as

```
= beta^D4*(1/(1-tau))*D5^(1-tau)
```

This is complicated so lets consider it one piece at a time. Begin with

```
beta^D4
```

This means that beta is raised to the power of the number in cell D4. This makes use of the "naming" capability for constants in Excel and is equivalent to `B17^D4`. The number in cell D4 is two so this term becomes

$$\beta^2$$

which is the discount factor squared. `Beta` is defined in line 17 of the spreadsheet as .98. Also, since

$$\beta = \frac{1}{1+\rho}$$

we can infer that the discount rate, ρ, is equal to about .02.

Next consider the term

```
(1/(1-tau))*D5^(1-tau)
```

which can be rewritten as

$$\frac{1}{1-tau} D5^{1-tau}$$

and since the cell D5 contains consumption we can further rewrite this expression as

$$\frac{1}{1-\tau} C_t^{1-\tau}$$

which is the same as the utility function in Eq. (6) above. So, the cell D12 contains the mathematics

$$\beta^t \frac{1}{1-\tau} C_t^{1-\tau}$$

which is the discounted utility for period t. Also, the parameter `tau` of the utility function is defined in line 16 of the spreadsheet as being equal to 0.5.

In summary, line 12 of the spreadsheet is used to calculate the discounted utility in each period and then to sum those values so as to obtained the total discounted utility in cell L12. Thus the criterion function for the model is contained in line 12.

Next consider the constraints of the model. Begin with the expression for production which is illustrated in Figure 1.3 below.

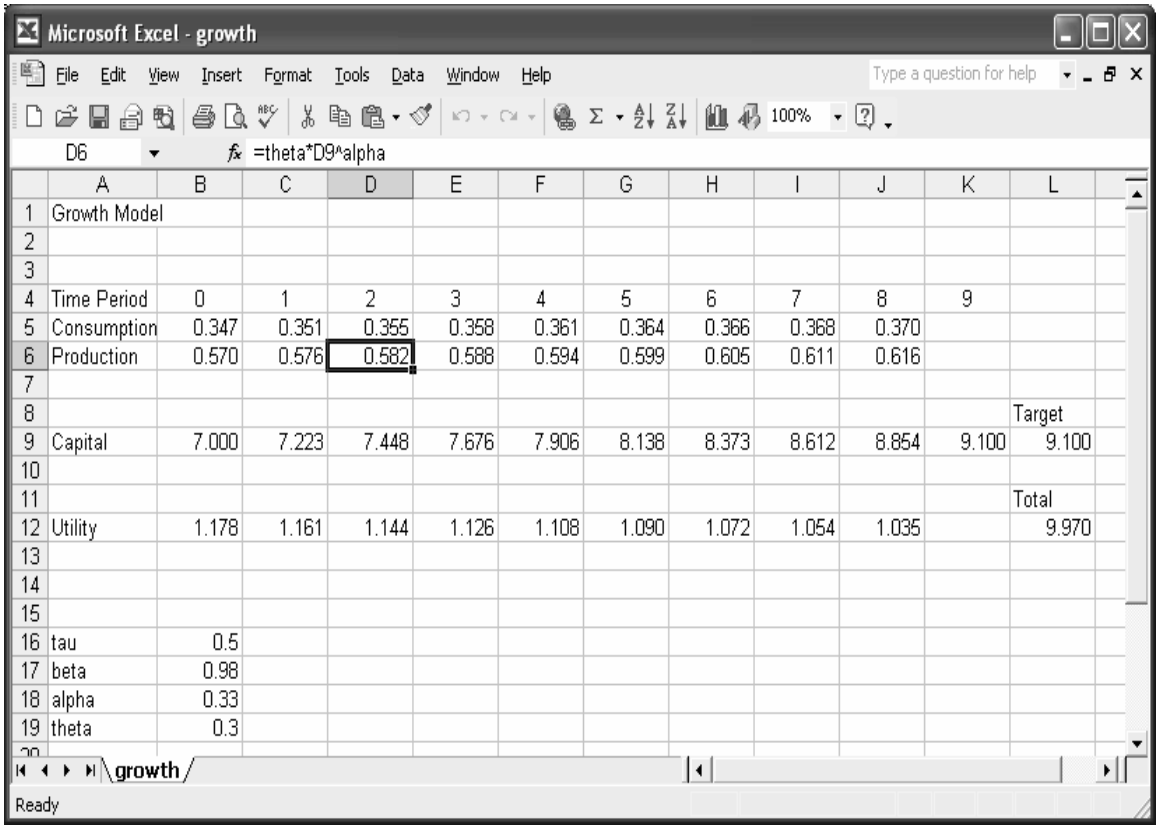


Figure 1.3 The Calculation of Production in Each Period

In this figure cell D6 is highlighted and the formula bar contains the expression

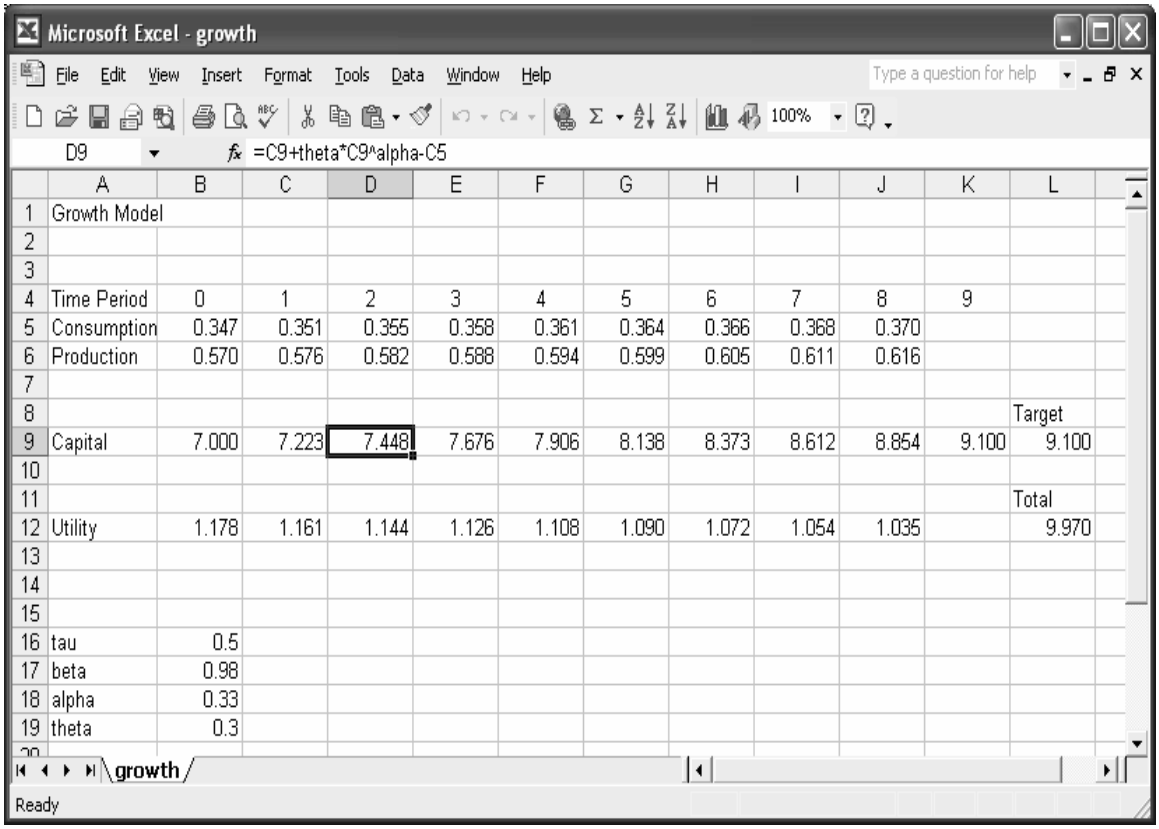
```
theta*D9^alpha
```

which is the same as Eq. (1) above for production, i.e.

$$Y_t = \theta \ K_t^{\alpha}$$

since cell D9 contains the capital stock for period t and `theta` is defined near the bottom of the spreadsheet in line 19 as being equal to 0.3 and `alpha` is defined in line 18 as being equal to 0.33.

Next consider the expression for the capital accumulation constraint which is shown in Figure 1.4 below where cell D9 is highlighted.



	A	B	C	D	E	F	G	H	I	J	K	L
1	Growth Model											
2												
3												
4	Time Period	0	1	2	3	4	5	6	7	8	9	
5	Consumption	0.347	0.351	0.355	0.358	0.361	0.364	0.366	0.368	0.370		
6	Production	0.570	0.576	0.582	0.588	0.594	0.599	0.605	0.611	0.616		
7												
8												Target
9	Capital	7.000	7.223	7.448	7.676	7.906	8.138	8.373	8.612	8.854	9.100	9.100
10												
11												Total
12	Utility	1.178	1.161	1.144	1.126	1.108	1.090	1.072	1.054	1.035		9.970
13												
14												
15												
16	tau	0.5										
17	beta	0.98										
18	alpha	0.33										
19	theta	0.3										

Figure 1.4 The Capital Accumulation Constraint

The expression in the formula bar this time, which is

```
C9 + theta*C9^alpha - C5
```

contains at its core the expression for production which we just developed above, i.e.

```
theta*C9^alpha
```

So we can translate the entire expression as

$$K_{t-1} + Y_t - C_{t-1}$$

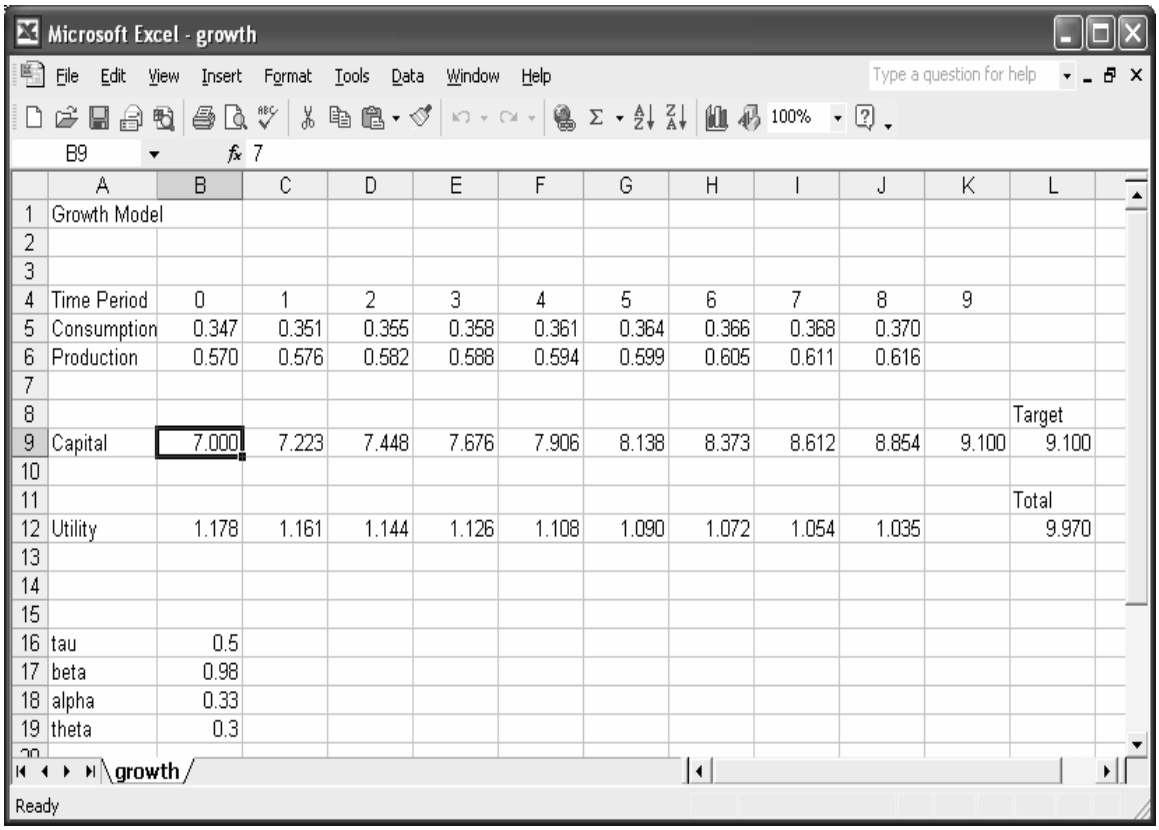
since row 9 contains the capital stock figures and row 5 contains the consumption figures. As you can see, by comparing the expression above to the capital accumulation constraint in Eq. (2) above with the time periods each decreased by one period, i.e.

$$K_t = K_{t-1} + Y_{t-1} - C_{t-1}$$

the timing in the spreadsheet calculations is slightly off for production but that slight timing error may make the spread slightly easier to specify so we will leave the error for the time being.

Also, notice at the end of row 9 in the spreadsheet that there is a target capital stock. We will discuss this in detail when we describe how the model is actualy solved

in Excel. However, before we do that it is necessay to indicate how the initial condition for capital stock is specified. This is shown below in Figure 1.5 where cell B9 is highlighted.



	A	B	C	D	E	F	G	H	I	J	K	L
1	Growth Model											
2												
3												
4	Time Period	0	1	2	3	4	5	6	7	8	9	
5	Consumption	0.347	0.351	0.355	0.358	0.361	0.364	0.366	0.368	0.370		
6	Production	0.570	0.576	0.582	0.588	0.594	0.599	0.605	0.611	0.616		
7												
8												Target
9	Capital	7.000	7.223	7.448	7.676	7.906	8.138	8.373	8.612	8.854	9.100	9.100
10												
11												Total
12	Utility	1.178	1.161	1.144	1.126	1.108	1.090	1.072	1.054	1.035		9.970
13												
14												
15												
16	tau	0.5										
17	beta	0.98										
18	alpha	0.33										
19	theta	0.3										

Figure 1.5 The Initial Capital Stock

When cell B9 is highlighted the fomula bar does not show a mathematical expression like those shown in the other cells in line 9, but rather just the number 7. This is the initial capital stock which was specified in the mathematical statement of the models in Eq. (4) as

$$K_0 \quad \text{given}$$

So the initial capital stock is given and it has been specified as equal to 7 in this version of the model.

The tour of the model in Excel given above is slightly confusing because it discusses both the data elements which the user must provide and the variables which are calcuated when the spreadsheet is solved. Now lets separate the two by looking again at Figure 1.5. The user must supply the time period numbers in row 4, the initial capital

stock in cell B9 and the parameter values `tau`, `beta`, `alpha` and `theta` in cells B16 through B19. Also, the user must supply the terminal capital stock target in cell L9. Excel will compute all the rest. Then why are all those other numbers shown in Figure 1.5? Those other numbers have all been computed the last time the model was solved and will be updated if you alter one of the inputs mentioned above and then solve the model again.

So lets consider next how the model is solved. This is accomplished by selecting the Tools menu and the Solver option from that menu. When you do this the dialog box in Figure 1.6 will appear.[5]

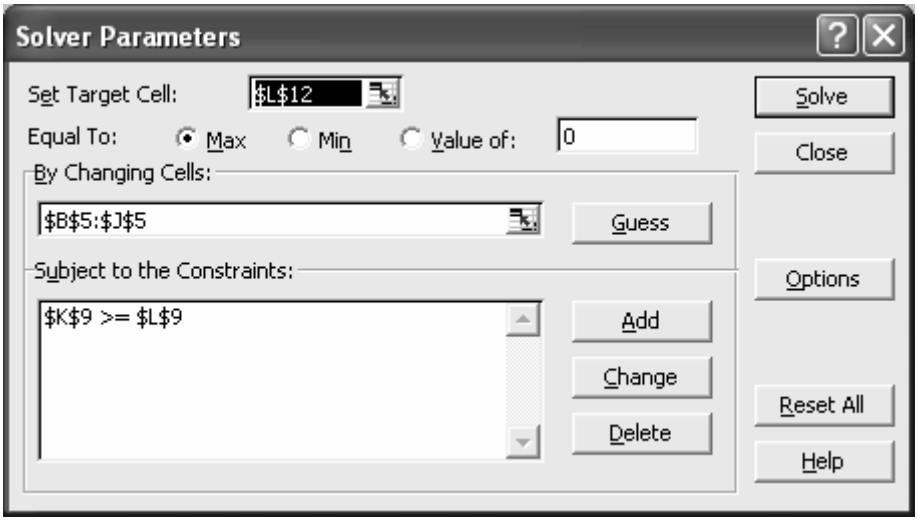


Figure 1.6 Solver Dialog Box

Consider first the top line in this dialog box in the section called "Set Target Cell". The edit box to the right of this capiton indicates that cell L12 has been chosen. This corresponds to the total discounted utility on the right hand side of the utility line in the spreadsheet. Just beneath this the user can specify whether the value in the cell is to be maximized or minimized. In the growth model at hand we are seeking to maximize the total discounted utility so "Max" is selected.

The next line is used to specify which cells are to be changed while seaching for the solution to the model. In the growth model we are solving for the values of consumption in each period that provide the best trade off between utility in that period and saving which becomes future capital stocks and permits more production later.

[5] In case the dialog box does not appear, see Appendix C.

Therefore, we specify here that the variables to be used in search for the optimum are those in cells B5 to J5 which are the consumption values.

Next consider the box that is labelled "Subject to the Constraints" in which appears the constraint

```
K9 >= L9
```

Since cell K9 contains the capital stock for period 9 and cell L9 contains the target capital stock, this constraint requires that the terminal period capital stock which is computed by the model be greater than or equal to the user specified target which in this case is set to 9.1, that is 30% higher than the intitial capital stock. This corresponds to the mathematical constraint in Eq. (5) above, i.e.

$$K_N \geq K^*$$

where K_N is the capital stock in the terminal period and K^* is the target capital stock.

Notice that it is not necessary in the Solver dialog box to specify all of the capital accumulation constraints in line 9 of the spreadsheet as constraints. Rather they are effectively linked together by the mathematical expressions so it is necessay to include only cell K9 when specifying the constraints.

To solve the model one selects the Solve button in the Solver dialog box in Fig. 1.6. What happens behind the scenes in the Excel program next is the solution of the nonlinear programming model that is represented by the growth model. A Newton method or a conjugate gradient method can be used in Excel to solve the model. A brief discussion of nonlinear optimization methods is provided in App F at the end of the book.

Clicking on the Options button in the Solver dialog box will display the Solver Options dialog box shown in Figure 1.7.

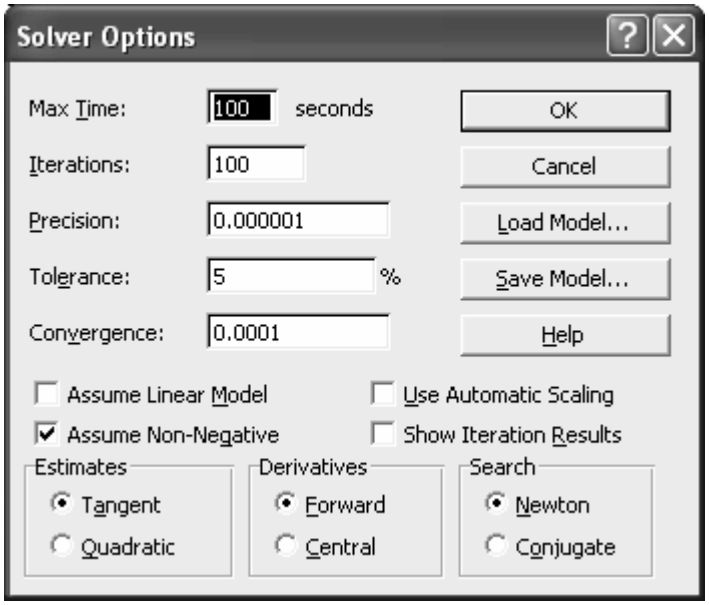


Figure 1.7 Solver Options Dialog Box

In this dialog box you will be able to change different parameters - i.e. maximum time, number of iterations, precision, tolerance and convergence - that allow you to control the performance of the nonlinear optimization method used by Excel. Notice that the Assume Non-Negative option has been selected to constrain the solution values of the model to non-negative values.

3. Results

When solving the growth model with the Excel Solver it is useful to remember the essential tradeoff in the model. More consumption today means more utility today. However, less consumption today means more saving and more investment today and this means more capital stock in the future and therefore more output and more consumption possibilities in the future.

So the problem is to find just the right level of consumption in each time period given the parameters of the model. The key parameters of the model are

β, beta	discount factor	0.98
K^*	target capital stock	9.1
θ, theta	production function parameter	0.30
α, alpha	production function exponent	0.33
K_0	initial capital stock	7
τ, tau	utility function parameter	0.50

The discount factor is the most intuitive of these parameters. Recall that is is equal to

(9) $$\beta = \frac{1}{1+\rho}$$

Solving Eq. (9) for the discount rate, ρ, yields

(10) $$\rho = \frac{1}{\beta} - 1$$

So when $\beta = .98$

(11) $$\rho = \frac{1}{0.98} - 1 = 1.02 - 1.00 = .02$$

and when $\beta = .95$

(12) $$\rho = \frac{1}{0.95} - 1 = 1.052 - 1.00 = .052$$

So to a reasonable approximation in the range of interest

(13) $$\beta \approx 1.00 - \rho$$

Thus a discount rate of six percent or .06 implies a discount factor of 0.94.

Next consider that the criterion function in Eq. (8) includes the discount factor, β, raised to the power t, i.e.

$$J = \sum_{t=0}^{N} \beta^t \frac{1}{(1-\tau)} C_t^{(1-\tau)}$$

and consider how β^t varies with beta and t as shown in Table 1.1.

		Values of β^t Corresponding to Each Time Period					
ρ	β	Time Periods					
		0	1	2	3	4	5
.02	.98	1.00	0.98	0.96	0.94	0.92	0.90
.05	.95	1.00	0.95	0.90	0.86	0.81	0.77

Table 1.1 Values of Discount Term

Thus when the discount rate is 5 percent the term β^t becomes smaller much faster as the time period increases than it does when the discount rate is 2 percent. So when the discount rate is higher, future utility is "discounted" more heavily, i.e. given less weight in the criterion function. Thus, if your discount rate is 2 percent you have relatively more interest in your consumption in future years than if your discount rate is 5 percent.

Therefore, altering `beta` is one of the interesting experiments to do with this model. As you increase the discount rate (and therefore decrease the discount factor beta) you should expect to see more consumption early in the time horizon covered by the model. An illustration of this result is shown in Figure 1.8 that contains plot lines for three experiments corresponding to three different values of `beta`.

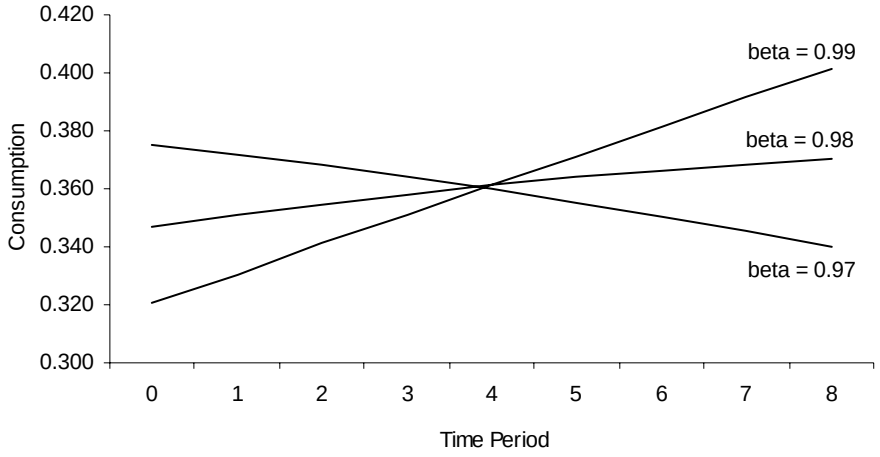


Figure 1.8 Consumption Paths for Different Discount Factors

A second key parameter of the model also plays a role in this matter of time preference of consumption and may affect your results in the experiments described above. This parameter is the target capital stock, K^*. The relevant constraint of the model is Eq. (5), i.e.

$$K_N \geq K^*$$

which requires that the capital stock in the terminal period exceed the target. This can be thought of as a constraint which represents the interest of the next generation. Without such a constraint, the optimal solution to the growth model will be to invest little or nothing in the last years covered by the model and to make consumption very high in those periods. So a constraint of this sort is normally added to numerical growth models.

There can be an interplay between the choice of discount rate and the choice of the target capital stock. If you choose a high target capital stock, then changes in the discount rate may not have much effect on the pattern of consumption over time since consumption must in any event be very low in order to insure that there is enough investment that the target capital stock can be met in the terminal period.

One of the most straightforward experiments with the model is to increase the initial capital stock. This has the effect of permitting more consumption with less investment and one would expect to see higher levels of both output and consumption in the model solution.

If you alter the θ parameter in the production function in Eq. (1), i.e.

$$Y_t = \theta \ K_t^{\alpha}$$

you are modifying the efficiency of the production process. For example, if you increase θ more output can be produced with the same capital stock and you should find higher levels of both output and consumption in the model solution. Similarly altering the α parameter affects the efficiency of the production process.

The last parameter that can be modified is τ - a parameter in the utility function. Intuition here is a little hard to come by, but as τ approaches zero the utility function becomes linear and as τ approaches one it becomes logarithmic so it may be useful to think of τ as a parameter which affects the curvature of the utility function or the degree of diminishing marginal utility.

Notice that when you perform these experiments, if the changes you make in the parameter values are relatively small, the Excel solver will easily converge to a new solution. This may not be the case for significant changes. Thus in those model runs you may have to "guess" and provide new values for the sequence of consumption values to be used by the Excel solver as new starting values, or you may have to play with different Solver Options to control the solver performance.

In contrast to numerical growth models, theoretical growth models are usually solved for infinite horizons and do not have a terminal capital stock target. As an approximation to this, some numerical growth models are solved for much longer time horizons than the period of interest and the solution is used only for a shorter period. Thus if one is interested in a twenty year period the model might be solved for forty or sixty years so that the end conditions do not have much effect on the solution paths for the first twenty years. When extending the time horizon, make sure that as you insert more columns to the Excel spreadsheet the equations of the model are copied in a proper maner, that the cell containing the sum of utilities is properly updated to cover the new range and that the specifications of the target cell, changing cell and the constraint are properly updated in the Solver dialog box.

An interesting experiment is to impose a terminal capital stock equal to the initial capital stock and solve the model for different time horizons. The optimal capital stock path for an experiment like this is shown in Figure 1.9.

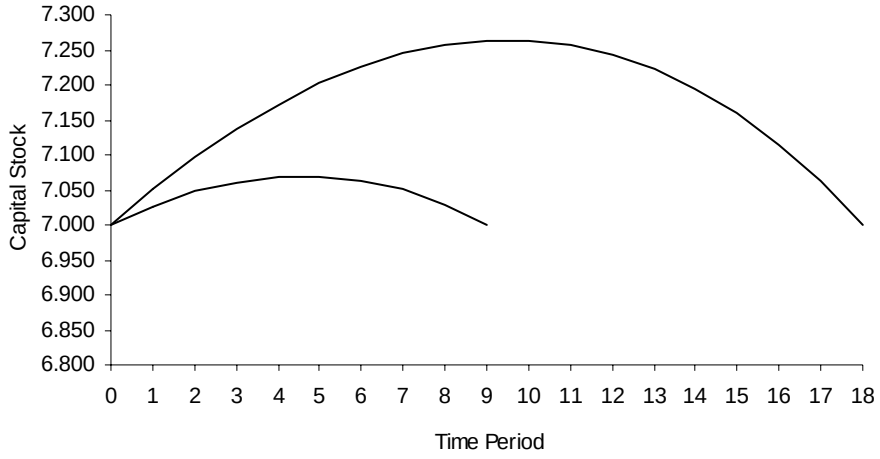


Figure 1.9 Capital Stock Paths

We can see that optimal values for the capital stock first increase then decrease. If we keep extending the time horizon, we will generate a sequence of even higher arches whose top parts will be flatter as they get closer to an upper limit value of about 10.5. This behavior is known at the "turnpike property". To understand this, we have to point out that a model like the one presented in this chapter has a steady-state solution, a solution that, given enough time, the consumption and capital stock levels would converge to and stay there forever. It can be shown (see Appendix J) that for this model the steady state capital stock is

$$K_{ss} = \left(\frac{1-\beta}{\beta \alpha \theta} \right)^{\frac{1}{\alpha - 1}} \tag{14}$$

Substituting the corresponding parameter values we obtain $K_{ss} = 10.559$. (To confirm that this is indeed a steady-state solution, you may want to impose this value as the initial and target capital stock values and solve the model with the Excel solver). Thus, any finite optimal path will tend to reach the steady state value, stay there or close to it as long as possible, and then leave it to go back to the target capital stock.

4. Experiments

Computational economics is not a subject that is easy to learn with the traditional lecture and exam style of teaching. Rather the crucial learning process is to first solve the models that other scholars have used, then to repeatedly make minor modifications and solve the model again in order to gain a clear understanding of how the model works and its strengths and weaknesses. At a later stage substantial structural changes can be made to the model so that it is more applicable to an economic situation of interest to the student.

Perform a series of experiments by modifying one of the parameters discussed above and observing the effects on the paths for capital stocks, output and consumption. Though it might be interesting to change more than one parameter at a time it is usually better when you are first studying a model to only change one parameter at a time. Save your results from one run to the next so that you can use Excel to plot the results across runs as in Figures 8 and 9.

A more challenging experiment that you may want to undertake (or may not want to undertake at this stage) is to treat the technology parameter θ, as stochastic. For example, you can define it as having a uniform distribution. To do so, you can use the Excel function RAND, which generates random numbers uniformly distributed between zero and one. Be aware that you should generate a random number for each time period.

5. Further Reading

Jones (1998) provides a systematic introduction to growth models. Azariadis (1993) and Barro and Sala-i-Martin (1995), at a more advanced level, present a variety of optimal growth Ramsey type models similar to the one developed in this chapter. Aghion and Howitt (1997) present a systematic treatment of endogenous growth models. Ros (2001) develops a presentation of growth models for developing countries. Mercado, Lin and Kendrick (2003) present a GAMS version of a single-sector growth model like the one used in this chapter and a multi-sector optimal growth model in GAMS that is an extension of the Kendrick and Taylor (1971) model. See Judd (1998) Ch. 13 for perturbation methods of solving growth models.

Chapter 2

Neural Nets in Excel

Much of economics is about finding optimal variables given parameters which describe human behavior. For example in the optimal growth model that we solved with Excel the goal was to find the optimal levels of the consumption and capital stock variables given the parameters of the production function and the utility function.

In this chapter we invert this duality. We begin with the observed behavior and attempt to find the parameters which permit the specified relationships to most closely fit the data. Such is the subject matter of econometrics and estimation. However, we will be looking at a type of estimation that has not been in the mainstream of econometrics but that developed in other fields and is now increasingly being used to fit economic relationships - namely neural nets.

Neural networks models are suitable to deal with problems in which relationships among variables are not well known. Examples are problems in which information is incomplete or output results are only approximations, as compared to more structured problems handled for example with equation-based models. Neural networks are particularly useful to deal with data sets whose underlying nonlinearities are not known in advance.[6] Among the many possible applications are forecasting and identification of clusters of data attributes.

The example we will use here is typical of the applications of neural nets to economics and finance - how best to predict the future prices of a stock.[7] The stock we use is that of the Ford Motor Company. We attempt to predict it by using the share price of a group of related companies - companies that provide inputs to automobile production and companies that produce competing vehicles.

The central notion of neural net analysis is that we can use a set of observations from the past to predict future relationships. Thus we use the closing price of Ford stock each week over a fourteen week period to "train" the model and then use the parameters

---

[6] One of the strengths of neural net methods is that they may approximate any functional shape.

[7] Neural nets are not necessarily a better way to predict stock prices than standard econometric methods; however stock prices offer a clear and motivating example for many students, thus we use that example here.

which emerge from the training to predict the Ford stock price in the fifteenth and sixteenth week. This is done in an Excel spreadsheet using the Solver that we first used in the growth model.

The chapter begins with an introduction to neural nets followed by the specification of an automobile stock price model. Then we will introduce the data that is used in the model, the representation of the model in Excel and the use of the Excel Solver to find the best parameter values.

1. Neural Nets Models

Neural networks (or, more properly, artificial neural networks) are inspired by, or up to a point analogous to, natural neural networks. They have three basic components: processing elements (called nodes or neurons), an interconnection topology and a learning scheme. From a computational point of view, a neural network is a parallel distributed processing system. It processes input data through multiple parallel processing elements, which do not store any data or decision results as is done in standard computing. As successive sets of input data are processed, the network processing functions "learn" or "adapt" assuming specific patterns which reflect the nature of those inputs.

There are many alternative network architectures. Let's look now in more detail at the elements, architecture and workings of a neural network as shown in Figure 2.1. This is known as backpropagation or as a feed forward model. This type of model is the most commonly used.

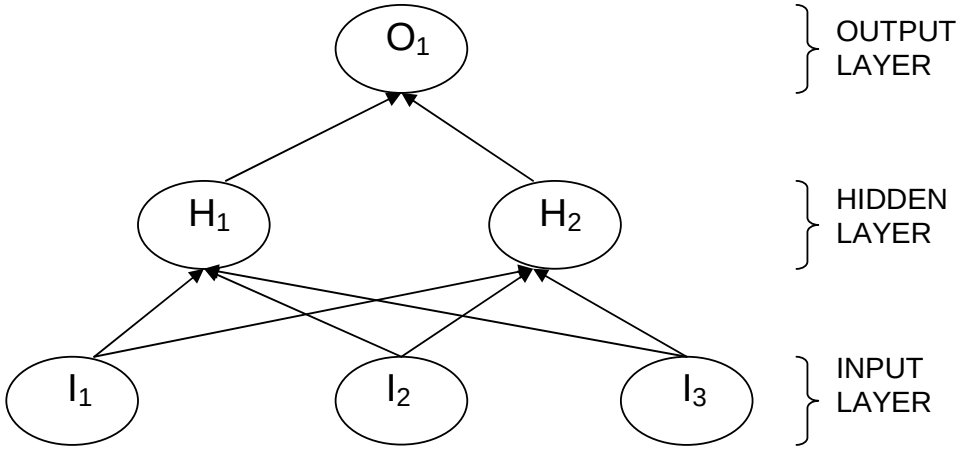


Figure 2.1 Neural Net Layers

This is a simple network with one input layer with three neurons, one intermediate layer with two neurons (usually named the "hidden layer") and one output layer with just one neuron. A key component of the network is the neuron, an elementary processing unit which generates output given inputs. It is composed of two main parts: a combination function and an activation function (Figure 2.2). The combination function computes the net input to the neuron, usually as a weighted sum of the inputs. The activation function is a function that generates output given the net input.

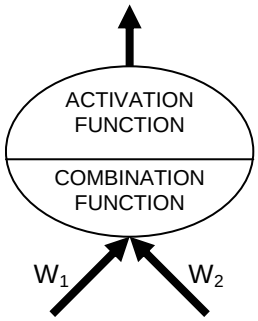


Figure 2.2 Activation and Combination Functions

It is standard procedure to constrain the output of a neuron to be within an interval (0,1). To do so, different functional forms can be used for the activation function, such as logistic functions, sigmoid functions, etc. Also, a threshold may be used to determine when the neuron will "fire" an output as the activation function yields a value above that threshold. Input layer neurons receive data ("signals") from outside and in general transmit them to the next layer without processing them. Output layer neurons return data to the outside, and are sometimes set to apply their combination functions only.

The learning process of the network consists of choosing values of the weights so as to achieve a desired mapping from inputs to outputs. This is done by feeding the network with a set of inputs, comparing the output (or outputs, in case of having more than one network output) to a known target, computing the corresponding error and sometimes applying an error function. Then weights are modified to improve the performance. To do this, a variety of methods can be employed, such as the Newton method or the conjugate gradient methods in Excel that are to be discussed later in this chapter.

2. The Automobile Stock Market Model

We begin with the specification of the combination function for the output layer as

$$y_t = \theta_0 + \sum_{j=1}^{q} \theta_j a_{tj} \tag{1}$$

where y_t is the output in period t, a_{tj} is the hidden node value in period t for node j and the θ_j's are parameters. There are q hidden nodes. In our model the y_t variables will be the share price of the Ford Motor Company stock in each of the fourteen weeks in 1997.

The θ's are among the parameters which we are seeking to find. The a_{tj}, which are the values in time period t at hidden node j, are given by the expression

$$a_{tj} = S\left(\sum_{i=1}^{q_j} w_{ji} x_{it} \right) \tag{2}$$

where the x_{it} are the inputs at node i in period t . There are q_j inputs at hidden node j.

The x_{it} are the share prices of the other companies in our example. The w_{ji} are the parameters at the jth hidden node for the ith input and are the second set of parameters that we are seeking to choose. Thus, in summary, we are given the share prices of the other companies x_{it} and the share price of the Ford stock y_t and are seeking to find the parameters θ and w which permit our functions to most closely fit the data.

What functions are being used? The first function in Eq. (1) is a linear function and the second function in Eq. (2), the function S, is a sigmoid function. The mathematical form of this function is

$$S(z) = \frac{1}{1+e^{-z}} \tag{3}$$

One can quickly see by examination that this function evaluated at $z = 0$ is

$$S(0) = \frac{1}{1+e^{-0}} = \frac{1}{1+1} = \frac{1}{2} \tag{4}$$

and that large negative values of z map to near zero, i.e.

$$S(-5) = \frac{1}{1+e^{5}} = .007 \tag{5}$$

and that large positive values of z map to approximately one, i.e.

$$S(5) = \frac{1}{1+e^{-5}} = .993 \tag{6}$$

So the function has the shape shown in Figure 2.3 below.

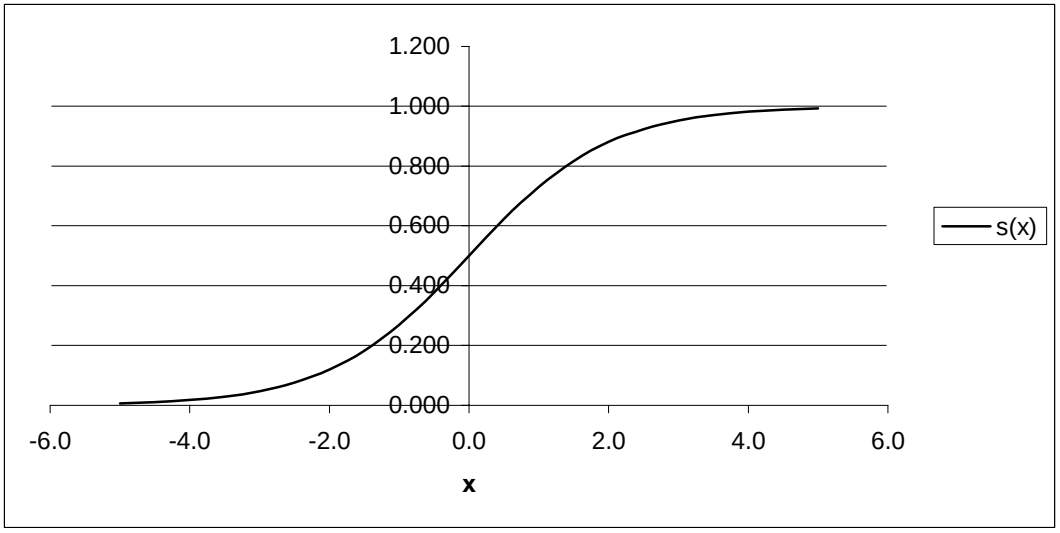


Figure 2.3 The Sigmoid Function

This function is sometimes called the "squasher" and it is quickly apparent why. Given any data set of numbers which range from very large negative numbers to very large positive numbers this function will map those numbers to the zero-to-one interval while maintaining their relative sizes.

The example we present here was developed by Joe Breedlove. This example contains share prices from the automotive suppliers of Ford in 1997, i.e.

Bethlehem Steel
Owen's Glass
Goodyear Tire and Rubber

and the competing auto makers to Ford, i.e.

Chrysler
General Motors

to predict the share price of the

Ford Motor Company.

At that time stock prices were quoted as fractions rather as decimals and the data in the spreadsheet reflect this fact. Also, the suppliers and competitors for the Ford Motor Company have changed since 1997; however the example is useful as a starting place for learning about neural nets.

The effect from the suppliers is aggregated into one hidden node and the effect from the competitors is aggregated into the second hidden node as is shown in Figure 2.4.

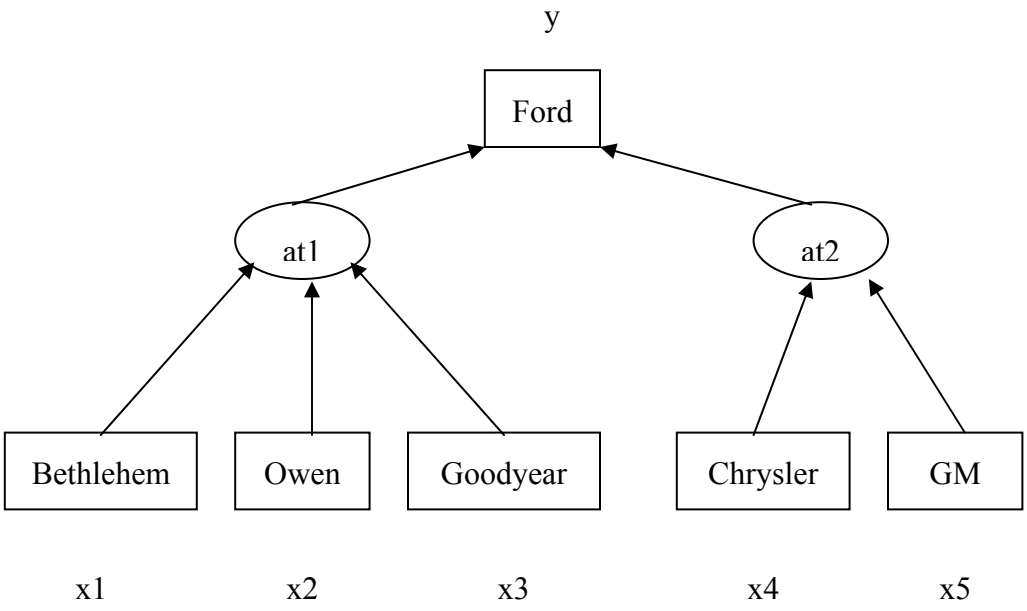


Figure 2.4 A Neural Net for Ford Motor Company Share Prices

So for the example at hand

(7) $$z_1 = w_{11} * x_1 + w_{12} * x_2 + w_{13} * x_3$$

and

(8) $$a_{t1} = \frac{1}{1 + e^{-(w11*x1 + w12*x2 + w13*x3)}}$$

(9) $$z_2 = w_{21} * x_4 + w_{22} * x_5$$

and

(10) $$a_{t2} = \frac{1}{1 + e^{-(w21*x4 + w22*x5)}}$$

Also

(11) $$\hat{y}_t = \theta_0 + \theta_1 a_{t1} + \theta_2 a_{t2}$$

Thus the optimization problem in Excel is to find the values of

(12) $$w_{11}, w_{12}, w_{13}, w_{21}, w_{22}, \theta_0, \theta_1, \theta_2$$

Which minimize the square of the separation between the predicted and actual values of the y's, i.e.

(13) $$Norm = \sum_{t=1}^{n} \left(y_t - \hat{y}_t \right)^2$$

where n is the number of observations which is fourteen for the example.

3. The Data

Closing stock prices for each week in the months of January, February and March of 1997 for Ford and for the three suppliers (Bethlehem, Owen and Goodyear) and the two competitors (Chrysler and GM) were used as shown in Table 2.1.

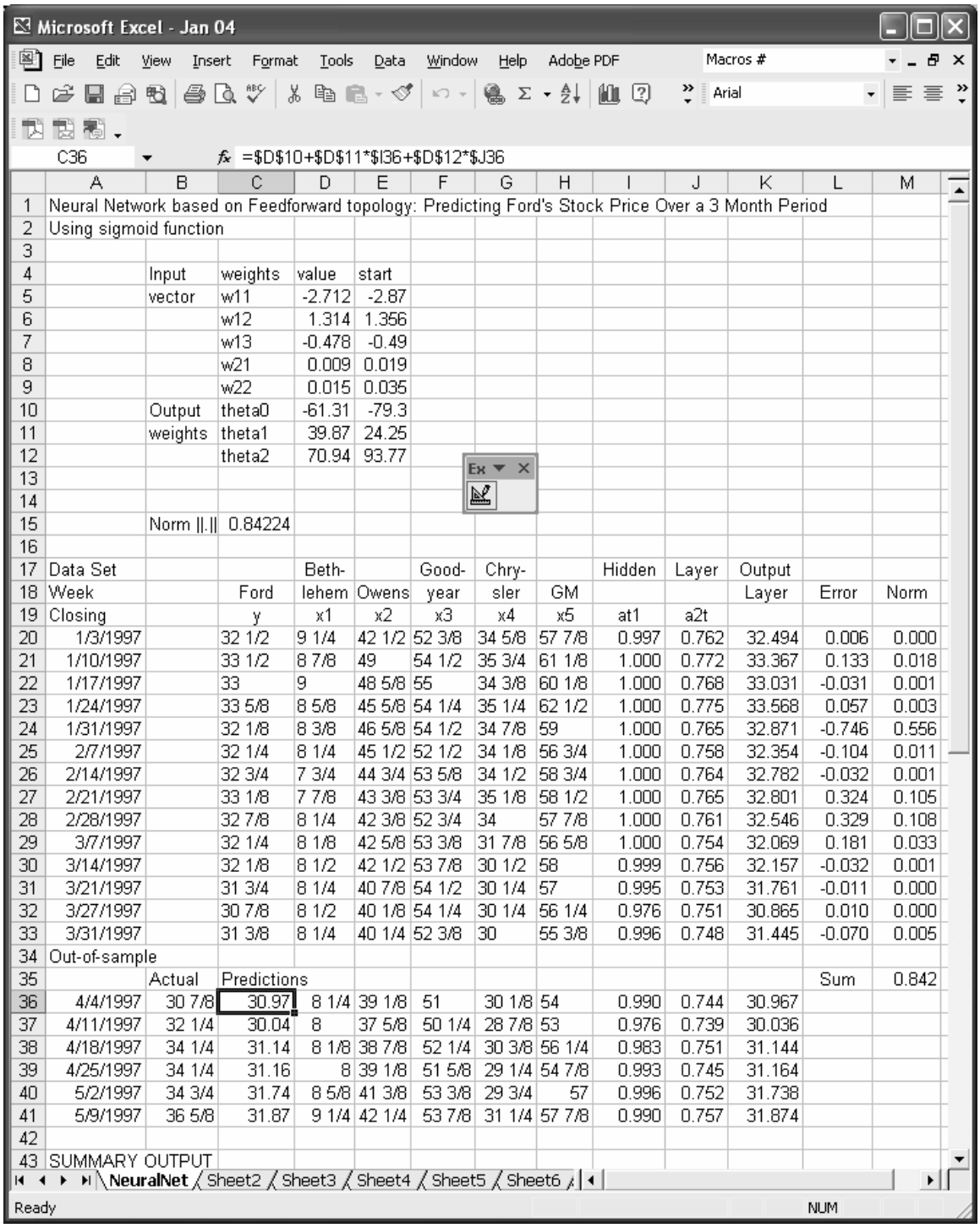
Week	Ford	Bethlehem	Owen	Goodyear	Chrysler	GM
Closing	y	x1	x2	x3	x4	x5
Jan 3	32 1/2	9 1/4	42 1/2	52 3/8	34 5/8	57 7/8
Jan 10	33 1/2	8 7/8	49	54 1/2	35 3/4	61 1/8
Jan 17	33	9	48 5/8	55	34 3/8	60 1/8
Jan 24	33 5/8	8 5/8	45 5/8	54 1/4	35 1/4	62 1/2
Jan 31	32 1/8	8 3/8	46 5/8	54 1/2	34 7/8	59
Feb 7	32 1/4	8 1/4	45 1/2	52 1/2	34 1/8	56 3/4
Feb 14	32 3/4	7 3/4	44 3/4	53 5/8	34 1/2	58 3/4
Feb 21	33 1/8	7 7/8	43 3/8	53 3/4	35 1/8	58 1/2
Feb 28	32 7/8	8 1/4	42 3/8	52 3/4	34	57 7/8
Mar 7	32 1/4	8 1/8	42 5/8	53 3/8	31 7/8	56 5/8
Mar 14	32 1/8	8 1/2	42 1/2	53 7/8	30 1/2	58
Mar 21	31 3/4	8 1/4	40 7/8	54 1/2	30 1/4	57
Mar 27	30 7/8	8 1/2	40 1/8	54 1/4	30 1/4	56 1/4
Mar 31	31 3/8	8 1/4	40 1/4	52 3/8	30	55 3/8

Table 2.1 Share Prices of Ford and Related Companies

As mentioned above, at that time stock prices were listed as fractional numbers, rather than as decimal numbers, as is now the case.

4. The Model Representation in Excel

Here we follow the representation of a neural net in Excel developed by Hans Amman and combine this with the model of Ford share prices of Joe Breedlove. The input file for Excel for this example can be obtained from the book web site. Once you have downloaded the file you can begin by opening it in Excel as is shown in Figure 2.5.



Microsoft Excel - Jan 04

File Edit View Insert Format Tools Data Window Help Adobe PDF Macros # Arial

C36 =$D$10+$D$11*$I36+$D$12*$J36

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Neural Network based on Feedforward topology: Predicting Ford's Stock Price Over a 3 Month Period												
2	Using sigmoid function												
3													
4		Input	weights	value	start								
5		vector	w11	-2.712	-2.87								
6			w12	1.314	1.356								
7			w13	-0.478	-0.49								
8			w21	0.009	0.019								
9			w22	0.015	0.035								
10		Output	theta0	-61.31	-79.3								
11		weights	theta1	39.87	24.25								
12			theta2	70.94	93.77								
13													
14													
15		Norm ‖.‖	0.84224										
16													
17	Data Set			Beth-		Good-	Chry-		Hidden	Layer	Output		
18	Week		Ford	lehem	Owens	year	sler	GM			Layer	Error	Norm
19	Closing		y	x1	x2	x3	x4	x5	at1	a2t			
20	1/3/1997		32 1/2	9 1/4	42 1/2	52 3/8	34 5/8	57 7/8	0.997	0.762	32.494	0.006	0.000
21	1/10/1997		33 1/2	8 7/8	49	54 1/2	35 3/4	61 1/8	1.000	0.772	33.367	0.133	0.018
22	1/17/1997		33	9	48 5/8	55	34 3/8	60 1/8	1.000	0.768	33.031	-0.031	0.001
23	1/24/1997		33 5/8	8 5/8	45 5/8	54 1/4	35 1/4	62 1/2	1.000	0.775	33.568	0.057	0.003
24	1/31/1997		32 1/8	8 3/8	46 5/8	54 1/2	34 7/8	59	1.000	0.765	32.871	-0.746	0.556
25	2/7/1997		32 1/4	8 1/4	45 1/2	52 1/2	34 1/8	56 3/4	1.000	0.758	32.354	-0.104	0.011
26	2/14/1997		32 3/4	7 3/4	44 3/4	53 5/8	34 1/2	58 3/4	1.000	0.764	32.782	-0.032	0.001
27	2/21/1997		33 1/8	7 7/8	43 3/8	53 3/4	35 1/8	58 1/2	1.000	0.765	32.801	0.324	0.105
28	2/28/1997		32 7/8	8 1/4	42 3/8	52 3/4	34	57 7/8	1.000	0.761	32.546	0.329	0.108
29	3/7/1997		32 1/4	8 1/8	42 5/8	53 3/8	31 7/8	56 5/8	1.000	0.754	32.069	0.181	0.033
30	3/14/1997		32 1/8	8 1/2	42 1/2	53 7/8	30 1/2	58	0.999	0.756	32.157	-0.032	0.001
31	3/21/1997		31 3/4	8 1/4	40 7/8	54 1/2	30 1/4	57	0.995	0.753	31.761	-0.011	0.000
32	3/27/1997		30 7/8	8 1/2	40 1/8	54 1/4	30 1/4	56 1/4	0.976	0.751	30.865	0.010	0.000
33	3/31/1997		31 3/8	8 1/4	40 1/4	52 3/8	30	55 3/8	0.996	0.748	31.445	-0.070	0.005
34	Out-of-sample												
35		Actual	Predictions									Sum	0.842
36	4/4/1997	30 7/8	30.97	8 1/4	39 1/8	51	30 1/8	54	0.990	0.744	30.967		
37	4/11/1997	32 1/4	30.04	8	37 5/8	50 1/4	28 7/8	53	0.976	0.739	30.036		
38	4/18/1997	34 1/4	31.14	8 1/8	38 7/8	52 1/4	30 3/8	56 1/4	0.983	0.751	31.144		
39	4/25/1997	34 1/4	31.16	8	39 1/8	51 5/8	29 1/4	54 7/8	0.993	0.745	31.164		
40	5/2/1997	34 3/4	31.74	8 5/8	41 3/8	53 3/8	29 3/4	57	0.996	0.752	31.738		
41	5/9/1997	36 5/8	31.87	9 1/4	42 1/4	53 7/8	31 1/4	57 7/8	0.990	0.757	31.874		
42													
43	SUMMARY OUTPUT												

NeuralNet Sheet2 Sheet3 Sheet4 Sheet5 Sheet6

Ready NUM

Figure 2.5 Spreadsheet for Neural Nets with Stock Prices

Skip down to the section on the data set beginning in line 17 and note that there are fourteen observations consisting of the weekly closing share price y for Ford shares

and the five inputs x1 through x5 for the other stocks. These observations are aggregated using the sigmoid function into the hidden layers at1 and at2 using a formula like

```
at1 = 1 / (1 + Exp(-(D20*D5 + E20*D6 + F20*D7)))
```

where the D5, D6 and D7 are weights that are to be solved for and the D20, E20 and F20 are the observations x1, x2 and x3. You can see this formula in the spreadsheet by selecting the I20 cell and then looking at the expression in the formula bar at the top of the spreadsheet. Alternatively, you can see all of the formulas in the spreadsheet by selecting

```
Tools:Options:Views
```

and then checking the

```
Formula
```

box.

Now back to the Data Set section of the spreadsheet. Check the column at2 and you will find that it is similar to the column at1 except that it uses data from the input data for x4 and x5 to compute the second of the two hidden layer values.

Consider next the Output Layer column. It is computed using an expression of the form

```
Output = theta0  +  theta1 * at1  +  theta2 * at2
```

where the thetas are weights which are computed in the optimization and that are shown in the section on Output weights near the top of the spreadsheet.

Next look at the Error column in the Data Set section of the spreadsheet. This column is simply the difference

```
Error = y - Output Layer
```

and the Norm column is the square of the elements in the Error column. The elements in the Norm column are summed up in cell M35 at the bottom of the column.

Now we are ready for the optimization problem. It is seen by selecting

```
Tools:Solve
```

and the following dialog box should appear.[8]

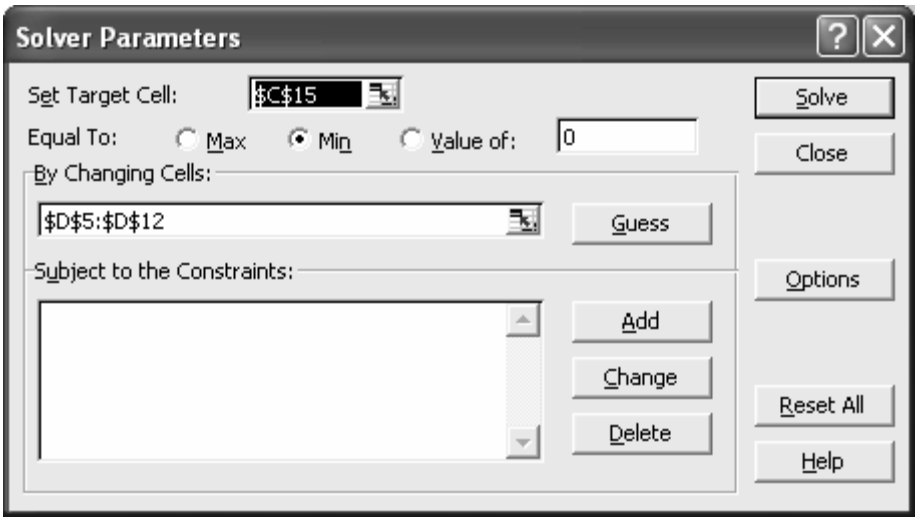


Figure 2.6 The Solver Dialog Box

This dialog box indicates the optimization problem is to minimize the value in cell `C15` (which on inspection is set equal to `M35` which in turn is the sum of the elements in the `Norm` column).

As was discussed earlier, the Excel Solver uses nonlinear optimization methods (Newton method or conjugate gradient method - see Appendix F). The optimization is done by changing the elements in the cells `D5:D12` until the minimum of the function is obtained. These cells are shown in Table 2.2 below beginning with the number -2.712 and going down the value column to the element 70.94.

[8] In case the dialog box does not appear, see Appendix C.

Input	weights	value	start
vector	w11	-2.712	-2.87
	w12	1.314	1.356
	w13	-0.478	-0.49
	w21	0.009	0.019
	w22	0.015	0.035
Output	theta0	-61.31	-79.3
weights	theta1	39.87	24.25
	theta2	70.94	93.77

Table 2.2 Parameters

The column to the right which is labeled `start` shows the numbers that were originally used when searching for the optimal parameters. They are not used in the present calculations but are stored there only to indicate which starting values were used. In fact each time the model is solved the numbers in the `value` column are used as the starting point and an effort is made to find values which will decrease the norm. So for a first experiment you might try changing some of the elements in the `value` column, selecting

```
Tools:Solver
```

and then clicking on the `Solve` button to solve the optimization problem and see if the parameters return to the original values or converge to some others which have either a smaller or larger norm.

A point of caution - at times the solution procedure will converge to a result with a higher norm because neural net estimation problems are sometimes characterized by nonconvexities and may have local optimal solutions that are not the same as the global optimal solution. Sometimes the number of local solutions may be very large. Thus in Excel it may be advisable to use a number of different starting values in order to check for global convergence. When there are many local optima global optimization algorithms such as genetic algorithms may be used to perform global exploration of the solution space – see the chapters on genetic algorithms or see Goldberg (1989).

Also, you can experiment by changing some data elements in the `y` and `x` columns either in an arbitrary manner or by looking up the share prices for these companies in another time period and seeing whether the parameter values have remained the same.

Finally the spread sheet contains some forecast in the section called `Predictions`. These predictions are made for six weeks after the last week for which

data was collected to 'fit' or 'train' the model. Look at the formulas for cells `B36` and `C36` that are shown in Table 2.3, which is shown below.

Out-of-sample		
	Actual	Predictions
4/4/1997	30 7/8	30.97
4/11/1997	32 1/4	30.04
4/18/1997	34 1/4	31.14
4/25/1997	34 1/4	31.16
5/2/1997	34 3/4	31.74
5/9/1997	36 5/8	31.87

Table 2.3 Predictions

If you select the cell just beneath the `Prediction` label you will see that the predictions use expressions like

```
= D10 +  D11*I36  +  D12*J36
```

that translates to

```
Output = theta0 + theta1 * at1 + theta2 * at2
```

Note in particular that these predictions are done from “out of sample” data, i.e. the data that is used to fit the model is not used to make the predictions. Rather some elements of the sample are reserved to test the model after it is fit to a subset of the data.

There is one other topic that needs to be mentioned about the Excel Solver. Select

```
Tools:Solver:Options
```

and the dialog box shown in Figure 2.7 will appear.

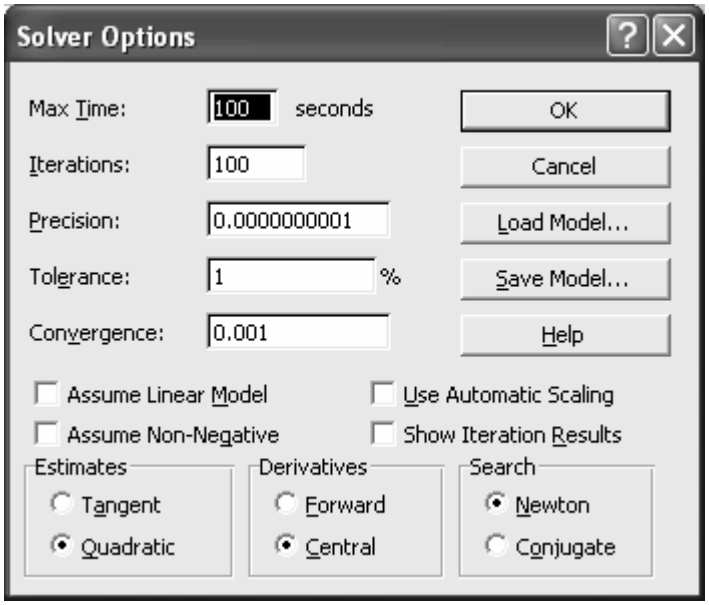


Figure 2.7 The Solver Options Dialog Box

You can use this dialog box to control the number of iterations which the Solver will use in trying to achieve convergence. Keep the number of iterations low when you are first working with a new data set and then if convergence is not being achieved raise this number as necessary. Also, a convergence value of 0.001 is probably close enough for most of the work you do, but you may require a looser convergence by lowering this setting to 0.01 in order to obtain convergence in 100 iterations. On the other hand you may want to keep the convergence value at 0.001 and increase the number of iterations.

Probably the most important element in the Solver Options Dialog Box is `Use Automatic Scaling`. In many neural net data sets the various series may be of very different magnitudes. For example you might have an unemployment series with numbers of the size of 0.04 and a consumption series with numbers like 625. In such a case it is wise to check the automatic scaling option. If you do this, the Solver will automatically scale all of your series so that they are roughly of the same magnitude and thereby increase the probability that the Solver will be able to find an optimal set of parameter estimates.

5. Experiments

There are two kinds of experiments which come to mind with this spreadsheet. As discussed above, at the simplest level you can change the data in the `y` and `x` columns and see how the weights and predictions change. You could even use your own data of some kind for doing this. Some students with greater interest in professional sports than in the stock market have used offensive and defensive statistics from basketball teams to predict the point spread in playoffs.

Also, you can change the number of input series `x1` thru `x5` by adding series such as `x6` and `x7` for other automotive companies such as Toyota and Honda. However, this is somewhat harder to do than the experiments discussed above since it involves making changes in the formulas in the spreadsheet. On the other hand this is a very good way to really learn how a neural net is represented and solved in a spreadsheet.

6. Further Reading

Sargent (1993) provides an introduction to neural nets. Garson (1998) presents an introduction to and a systematic coverage of the use neural networks in the social sciences. Beltratti, Margarita and Terna (1996) also present an introduction to neural networks and develop a variety of models for economic and financial modeling.

Chapter 3

Partial Equilibrium in Mathematica

It is customary to begin the study of microeconomics with market behavior in a partial equilibrium setting. This is done by analyzing the determination of price and quantity in a single competitive market under the assumption that all other influences from the rest of the economy remain constant. This study usually begins with the theory of the consumer and the derivation of demand curves and then proceeds to the theory of the firm and the derivation of supply curves. Following this demand and supply are brought together to study market equilibrium. This is the standard approach we will follow here. We will mainly be interested in the derivation of analytical results and graphical representations, something for which Mathematica is a very useful tool due to its power to deal with symbolic mathematics problems and to its plotting capabilities.

1. Utility and Production Functions

The starting point of consumer theory is the specification of preferences and their representations by means of a utility function, while the starting point of the theory of the firm is the specification of technology and its representation by means of a production function. While many theoretical results are derived for very general forms of those functions, in most examples, and also in applied work, it is common to work with a few functional specifications. Leontief and Cobb-Douglas functions are probably the most popular, and they can be used to represent preferences or technology. In the following we will present each of them. We will focus on the two-good case since this case can be easily handled in graphical representations, though the results can be generalized to more goods and the results displayed analytically.

1.1 Leontief Function

A Leontief function for a two good case is

(1) $$f(x_1, x_2) = \min(a_1 x_1, a_2 x_2)$$

where f is the function, a_1 and a_2 are parameters, and x_1 and x_2 are interpreted as goods consumed, if we use the function to represent preferences as in consumer's theory. Alternatively, they may be interpreted as inputs if we use the function to represent technology as in the theory of the firm. This function specifies that no substitution is possible between goods or between inputs. The consumer will always spend all of his or her income in fixed proportions between the two goods, and a similar behavior will be displayed by the firm in connection with its inputs. As we will see later, this will imply a peculiar form for the consumer's indifference curves and for the firm's production isoquants.

The graphical representation of a function like this in Mathematica is straightforward, and it is available in the `Leontief.nb` file in the book web site. You can begin with that notebook file if you are already somewhat familiar with Mathematica. Or if you are a first-time Mathematica user, we recommended that you type in the commands. The instructions for running Mathematica are in Appendix B.

We begin by assigning values to the parameters a_1 and a_2. In this case we assign the value 1 to both of them. Start Mathematica and on the Untitled-1 window that opens type

a1 = 1

followed by `Return` and then

a2 = 1

followed by `Shift-Enter` Mathematica acts as an interpreter and commands are processed one at a time. When you use Return at the end of the line you effectively ask Mathematica to postpone the processing while you proceed to enter another command on the next line. When you use `Shift-Enter` at the end of the line you ask Mathematica to process all of the input since the last `Shift-Enter`. Mathematica will then respond by converting your input to

```
IN[1]:=     a1 = 1
            a2 = 1
```

The symbols *IN[]:=* in Mathematica denote input and the other expressions are the input to be evaluated. The output statements corresponding to the input are

```
Out[1]:=    1
Out[2]:=    1
```

Thus Mathematica displays as output the result of the assignments. Notice that separate output is generated for each statement, no matter if we wrote the inputs in a single input prompt or in separate ones. Notice also the sequential numbering of inputs and outputs.

The outputs of the previous evaluations are quite simple and redundant. To avoid the display of output, we could have added a semicolon " ; " at the end of the statement whose output we wanted to suppress.

Next we assign to the variable Leontief the corresponding Mathematica function `Min[]` which yields the numerically smallest of its arguments.

```
IN[3]:=     Leontief = Min[a1 x1, a2 x2]
```

Notice that in Mathematica two symbols can be multiplied either by using the asterisk operator as `a1*x1` or simply by juxtaposing the two symbols with a space between them as `a1 x1`. When you finish typing the line above be sure to strike `Shift-Enter`. This will yield the output

```
OUT[3]:=    Min[x1,x2]
```

Notice that Mathematica replaced the parameters `a1` and `a2` with their numerical values of 1 while keeping everything else the same since the evaluation of the statement cannot be carried out, for the time being, beyond this point.

Next we ask Mathematica to generate a three-dimensional plot of the function within given numerical intervals for x_1 and x_2 using the Mathematica function `Plot3D[f,{x,xmin,xmax},{y,ymin,ymax}]` where `f` is the function to be plotted over the variables `x` and `y` between their specified minimum and maximum values. So type

```
IN[6]:=     Plot3D[Leontief,{x1,0,1},{x2,0,1}]
```

Be careful not to misspell "Leontief" or Mathematica will give you more error messages than you care to see. Also, be sure to end the line with `Shift-Enter`. The resulting graph is shown in Figure 3.1

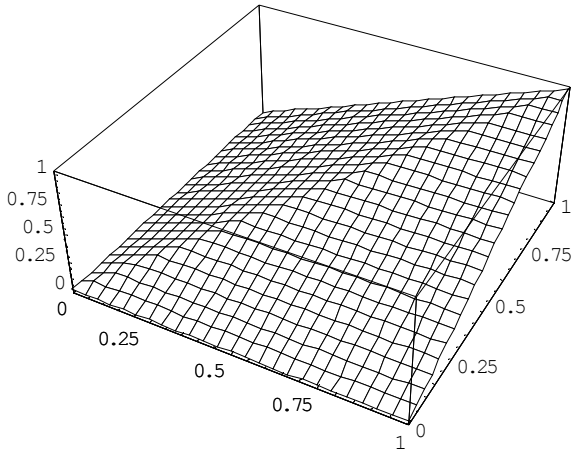


Figure 3.1 Leontief Function

Finally, with the statement

```
IN[8]:=     ContourPlot[Leontief,{x1,0,1},{x2,0,1}]
```

we obtain the contour plot of the Leontief function shown in Figure 3.2, which show us the consumer's indifference curves or, equivalently, the firm's isoquants. Contour plots produced by Mathematica are by default shaded, and regions with higher functional values are lighter. Contour curves for the Leontief function form ninety degree angles. Notice that the graph shows the kinks with some error as we get farther away from the origin.

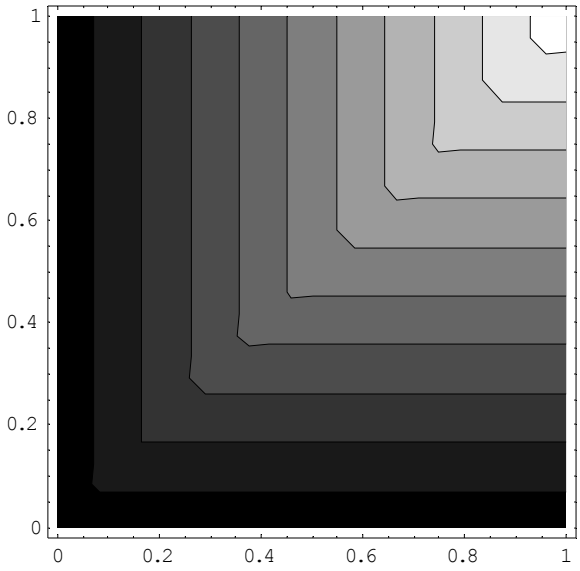


Figure 3.2 Leontief Function Contour Lines

Every time you run a program in Mathematica it is important to wipe out any previous values associated with the parameters and variables of the problem. This could have been achieved by adding the following statement at the beginning of the program

```
IN[]:=      Clear[a1,a2,x1,x2,Leontief];
```

1.2 Cobb-Douglas Function

A Cobb-Douglas function with constant returns to scale (we use a special case) is

$$f(x_1, x_2) = x_1^{\rho}\, x_2^{1-\rho} \tag{2}$$

where f is the function, x_1 and x_2 are goods or inputs, and ρ is a parameter. In consumer theory ρ and $1-\rho$ represent the consumer's expenditure shares on each good. In the theory of the firm, since the two exponents of the inputs add up to one, it implies that the technology the functions represent displays constant returns-to-scale. Unlike the Leontief function, this function allows for smooth substitution between goods or between inputs.

The Mathematica statements corresponding to the graphical representation of the Cobb-Douglas function are shown below and are available in the `CobbDouglas.nb` file in the book web site. This time we recommend that you open the input files and use it to

follow the discussion. When you open the notebook file you will see a bunch or brackets on the right hand side of the window. You can execute the program by selecting these brackets and striking Shift-Enter. For example selecting the bracket opposite the lines

```
Clear[x1,x2,ρ];
ρ = 0.7;
CD = x1^ρ  x2^(1-ρ);
Plot3D[CD,{x1,0,1},{x2,0,1}]
ContourPlot[CD,{x1,0,1},{x2,0,1}]
```

And striking Shift-Enter causes they lines to be processed and results in their being reprinted as

```
In[1]:=
      Clear[x1,x2,ρ];
      ρ = 0.7;
      CD = x1^ρ  x2^(1-ρ);
      Plot3D[CD,{x1,0,1},{x2,0,1}]
      ContourPlot[CD,{x1,0,1},{x2,0,1}]
```

with input prompt `In[1]:=` now showing. In this way you can use the notebook files to modify the input and rerun the program. For example you might have changed ρ from 0.7 to 0.8 and then selected the bracket to its right and type Shift-Enter. Be aware however, that only that part of the program covered by the bracket you selected will be rerun. Therefore if you want to redo the plots you must select one of the more inclusive brackets on the right before striking `Shift-Enter`.

The statements above follow the pattern presented in the previous section. We named the function `CD` and we have assigned a value of 0.7 to the ρ parameter. Unlike the program for the Leontief function, here we put all the statements together in one input prompt, and suppressed output using semicolons at the end of the first three statements. Notice that Mathematica allows you to enter Greek letter symbols like ρ. To do so, and also to enter formulas in a mathematical form instead of the text form we used here, you have to use a palette you can access from the File/Palettes/BasicInput main menu option.

Figures 3 and 4 show the corresponding three dimensional and contour graphs.

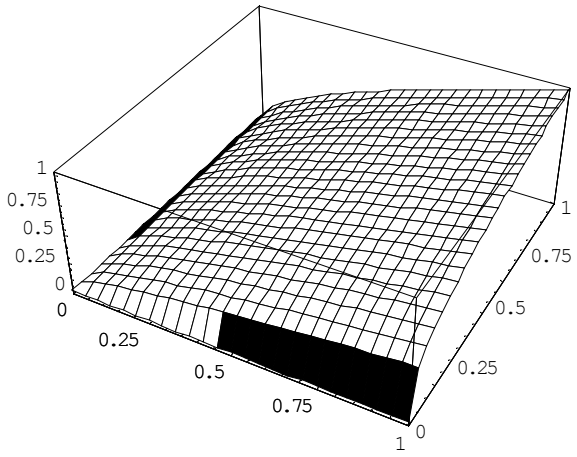


Figure 3.3 Cobb-Douglas Function

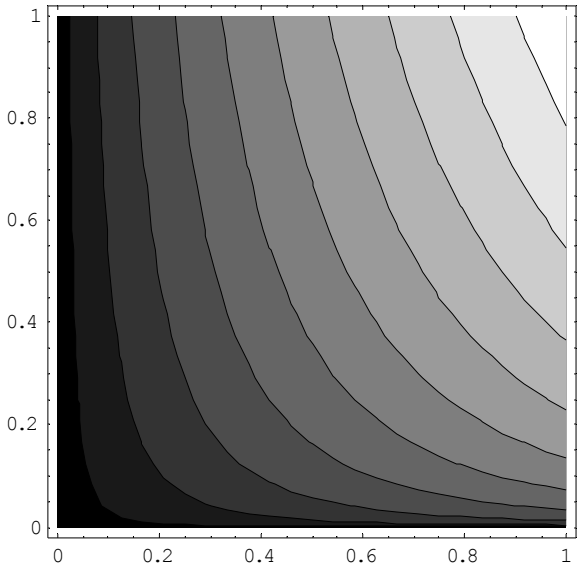


Figure 3.4 Cobb-Douglas Function Contour Lines

If you are following along with Mathematica, you might close all the files you have opened so far to reduce the clutter on your computer desktop and give yourself a fresh start in the next section.

3. Consumer Theory

The standard theory of consumer's behavior poses the problem faced by the consumer as one of maximizing utility subject to a budget constraint. That is, given a bundle of goods, their prices and a certain amount of income, the consumer buys those goods according to her preferences while trying to maximize her utility, a quantity that is supposed to measure the level of consumer satisfaction.

In formal terms, and for a two-good example that can be easily generalized, the problem can be stated as

$$\begin{aligned} &\max \quad u(x_1, x_2) \\ &\text{subject to} \quad p_1 x_1 + p_2 x_2 = m \end{aligned} \tag{3}$$

where u is the utility function, x_1 and x_2 are goods, p_1 and p_2 are prices and m is income.

From now on we will work with a Cobb-Douglas function. Thus, using (2) the problem above can be restated as

$$\begin{aligned} &\max \quad u = x_1^{\rho}\, x_2^{1-\rho} \\ &\text{subject to} \quad p_1 x_1 + p_2 x_2 = m \end{aligned} \tag{4}$$

An equivalent but simpler expression of the utility function is obtained taking logs

$$\log u = \rho\, \log(x_1) + (1-\rho)\, \log(x_2) \tag{5}$$

We start the Mathematica program of the consumer's problem - available in the `Consumer.nb` file - by inputting the utility function

```
In[]:=      logu = ρ Log[x1] + (1-ρ) Log[x2];
```

and the budget constraint

```
In[]:=      bc = m - (p1 x1 + p2 x2);
```

Notice that we give a name to the budget constraint then assign to it all its elements. We soon will see the usefulness of doing that.

The next step is to form the Lagrangian corresponding to the maximization problem. Thus we write

```
In[]:=      eqL = L == logu + λ bc
```

Notice that we assign to the variable `eqL` the expression `L == logu + λ bc`. The presence of the double equal symbol "==" indicates that the expression is an equation, not an assignment to the variable `L`. The corresponding output is the content of the variable `eqL` with the expressions for `logu` and `bc` being replaced by their definitions.[9]

```
Out[]=      L =(m - p1 x1 - p2 x2) λ + ρ Log[x1] + (1-ρ) Log[x2].
```

If instead of writing the budget constraint in the way we did above, we write it in a more standard way i.e.

```
In[]:=      m = p1 x1 + p2 x2;
```

to later write down the Lagrangian as

```
In[]:=      eqL = L == logu + λ (m - p1 x1 - p2 x2)
```

the output generated by Mathematica would be

```
Out[]=      L == ρ Log[x1] + (1-ρ) Log[x2]
```

Indeed, when evaluating the part of the input expression corresponding to `(m - p1 x1 - p2 x2)`, Mathematica will replace the variable `m` with its definition. Then this part of the expression would become `(p1 x1 + p2 x2 - p1 x1 - p2 x2)`. Thus, it would be equal to zero. It was to avoid this kind of problem that we defined the variable `bc` in the way we did above.

---

[9] It is common in Lagrangian functions to put the objective term first followed by the lambda and the constraint. However, given the sequence of commands we used, Mathematica does things in reverse order. This causes no problem except making the output below slightly harder to comprehend at first.

Once we form the Lagrangian, we compute the first order conditions of the problem as follows

```
In[]:=      foc1 = D[eqL, x1]
            foc2 = D[eqL, x2]
            foc3 = D[eqL, λ]
```

The Mathematica function `D` computes the partial derivatives of a function. In this case, we ask Mathematica to compute the partial derivatives of the expression `eqL` w.r.t. the variable of choice. The corresponding outputs are

$$\text{Out[]=} \quad 0 == -p1\,\lambda - \frac{\rho}{x1}$$

$$\text{Out[]=} \quad 0 == -p2\,\lambda + \frac{1-\rho}{x2}$$

$$\text{Out[]=} \quad 0 == m - p1\ x1 - p2\ x2$$

From the system of equations formed by the first order conditions we can obtain the goods' demand functions. The Mathematica function `Solve` allows us to do so. Within this function, we first have to specify the equations and then the variables over which they are solved.

```
In[]:=      Solve[{foc1,foc2,foc3},{x1,x2,λ}]
```

The previous statement generates the output

$$\text{Out[]=} \quad \left\{\left\{\lambda \to \frac{1}{m},\ x1 \to \frac{m\rho}{p1},\ x2 \to \frac{m - m\rho}{p2}\right\}\right\}$$

Finally, we want to plot the good's demand functions. Since the standard procedure is to plot quantities in the horizontal axis and prices in the vertical axis, we have to solve the demand functions for the corresponding prices. Starting with good 1, the Mathematica statements are

```
In[]:=      p1 = ρ m / x1;

            Plot[p1 /. {ρ → 0.7, m → 0.1},
            {x1,0.01,0.1},
            AxesLabel → {"x1", "p1"},
            PlotLabel → "Demand Curve for x1"]
```

In the first line of the `Plot[]` function the replacement operator "`/.`" is used. This operator, whose general form is "`expression /. rules`" applies a rule or list of rules in an attempt to transform each subpart of an expression. In our case the transformation rules are `ρ → 0.7` and `m → 0.1` which are used to give particular values to the parameters `ρ` and `m`. To write the arrows, you must type `->` as a pair of characters, with no space in between.

The second line of the `Plot` function contains the specification of the range for the horizontal axis, writing first the name of the corresponding variable then the minimum and the maximum values for the plot. Finally, the last two lines label the axes and assign the plot a label by means of the options `AxesLabel` and `PlotLabel`. The plot generated is shown in Figure 3.5.

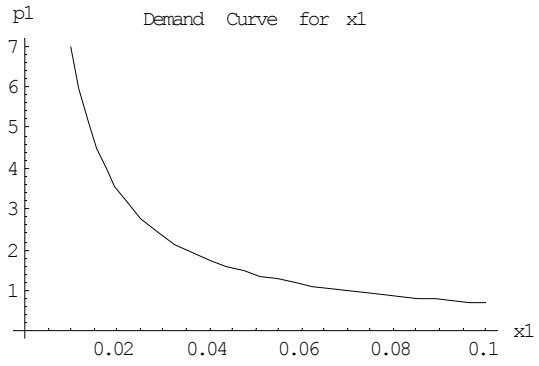


Figure 3.5 Demand Curve for x1

In an analogous way, we generate a plot for the demand function of good `x2` which is shown in Figure 3.6.

```
In[]:=    p2 = (m - ρ m) / x2;

          Plot[p2 /. {ρ → 0.7, m → 0.1},
          {x1,0.01,0.1},
          AxesLabel → {"x2", "p2"},
          PlotLabel → "Demand Curve for x2"]
```

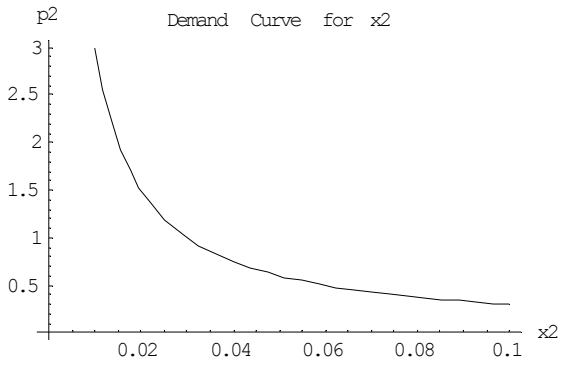


Figure 3.6 Demand Curve for x2

3. The Theory of the Firm

The standard theory of firm's behavior assumes that the main goal of the firm is to maximize profits given technology and the prices of output and inputs. To develop a simple example, let's assume that the firm produces a single output x_1 with price p_1, using labor L as a single input and whose price is the wage w. Let's assume also that the production function is of the form $T L^b$ where T and b are parameters and let's denote profits by π.

In formal terms the problem of the firm can be stated as

$$\begin{aligned} &\max \quad \pi = p_1 x_1 - wL \\ &\text{subject to} \quad x_1 = T L^b \end{aligned} \tag{6}$$

Substituting the production function into the profit function we obtain the first input for the Mathematica representation of the problem - available in the `Firm.nb` file - as

```
In[]:=      pi = p1 T L^b - w L;
```

Notice that we wrote `pi` instead of π since the Greek letter π is a reserved symbol in Mathematica.

Next we solve the first order condition of the problem for `L`. By means of the `D[ ]` function we compute the partial derivative of the profit function w.r.t. the variable labor then set the result equal to zero. Finally, we nest this operation within a `Solve[ ]` function.

```
In[]:=      Solve[D[pi,L]==0,L]
```

The resulting output is the labor demand function

```
Out[]=
```

$$\left\{\left\{L \to \left(\frac{w}{b\ p1\ T}\right)^{\frac{1}{-1+b}}\right\}\right\}$$

Next we assign the expression for the labor demand function to the temporary variable `tempL`. To do so we use the replacement operator "`/.`". The `%` symbol in the statement below refers to the last result generated, and `[[1]]` which refers to the first solution from the output list, which in this case contains only one solution. Thus, `tempL` will be equal to `L` where `L` is replaced by the solution generated in the previous output line.

```
In[]:=      tempL = L /.%[[1]]
```

Substituting `tempL` - that is, the labor demand function - into the production function in Eq. (6) we obtain the supply function for `x1` which we assign to the temporary variable `tempx1`.

```
In[]:=      tempx1 = T tempL^b
```

The resulting output is

```
Out[]=
```

$$T\left(\left(\frac{w}{b\ p1\ T}\right)^{\frac{1}{-1+b}}\right)^{b}$$

Having obtained the good supply and the labor demand functions, we want to plot them in the standard way, that is with price and wage in the vertical axis respectively. Begin with the good supply function. In the next two statements we (1) create an equation setting `x1` equal to the expression contained in the temporary variable `tempx1` and (2) assigning to the variable `plotx1` the result of solving the equation for `p1`.

```
In[]:=  eqx1 = x1 == tempx1;
        plotx1 = Solve[eqx1,p1]
```

The result is the inverted good supply function where `p1` appears as a function of `x1`.

```
Out[]=
```

$$\{\{p1 \to \frac{w\left(\left(\frac{x1}{T}\right)^{\frac{1}{b}}\right)^{1-b}}{b\ T}\}\}$$

Finally, we assign the result above to the temporary variable `tempp1`, give numerical values to the parameters and generate the corresponding plot, obtaining the graph shown in Figure 3.7.

```
In[]:=  tempp1 = p1 /. plotx1[[1]];
        Plot[tempp1 /. {b → 0.4, T → 1, w → 100} ,
        {x1,0.01,0.1},
        AxesLabel → {"x1", "p1"},
        PlotLabel → "Supply Curve for x1"]
```

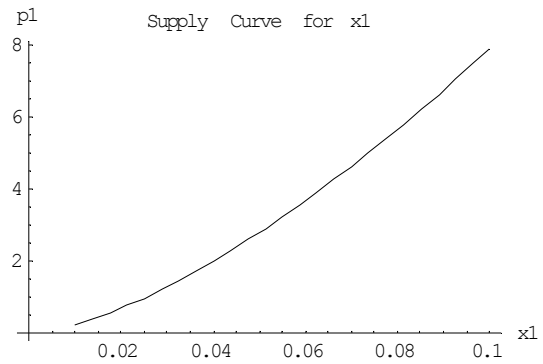


Figure 3.7 Supply Curve for x1

In a similar way, with the statements below we generate the plot for the labor demand curve shown in Figure 3.8.

```
In[]:=      eqL = L == tempL;
            plotL = Solve[eqL, w];
            tempw = w /. plotL[[1]]
            Plot[tempw /.  {b → 0.4, T → 1, p1 → 1},
            {L,0.01,0.1},
            AxesLabel → {"L", "w"},
            PlotLabel → "Labor Demand Curve"]
```

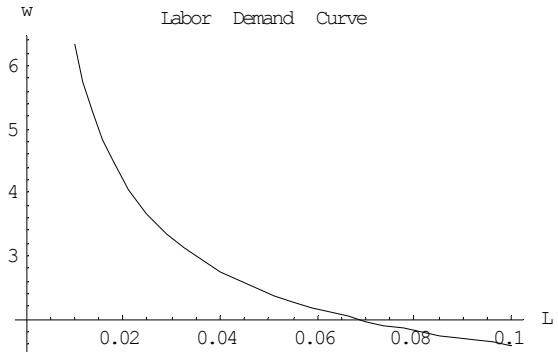


Figure 3.8 Labor Demand Curve

Now we are in a position to turn our attention to the market equilibrium.

3. Market Equilibrium

Having derived demand and supply curves, it is time to put them together to analyze the resulting market equilibrium. We will do so for the case of good `x1`. We begin from the corresponding demand and supply curves obtained in the previous sections with a slight modification: the variable `p1` from the demand curve will be renamed `p1d`, while the variable `p1` from the supply curve will be renamed `p1s`.

We begin the Mathematica representation of the model of partial market equilibrium - available in the `MarketEquil.nb` file - with the statements

```
In[]:=    p1d = ρ m / x1;
          p1s = w (((x1 / T)^(1 / b))^(1-b)) / (b T);
```

Then we solve for the equilibrium quantity when demand equals supply

```
In[]:=    equilx1 = Solve[p1d == p1s,x1]
```

obtaining as output

$$\text{Out[]=}\quad \{\{x1 \to \left(\frac{T^{-1/b} w}{b\, m\, \rho}\right)^{-b}\}\}$$

Then the equilibrium price can be obtained by substituting the solution for `x1` into `p1d`

```
In[]:=    equilp1 = p1d /. equilx1[[1]]
```

$$\text{Out[]=}\quad m \left(\frac{T^{-1/b} w}{b\, m\, \rho}\right)^{b} \rho$$

Next we assign values to the parameters and to the wage variable, and we compute the corresponding numerical values for the equilibrium quantity and price. To do so, we write the variables equilx1 and equlp1 without semicolons, since Mathematica

will automatically replace each parameter with its value and perform the corresponding calculations.

```
In[]:=      ρ = 0.7;
            m = 0.1;
            T = 1;
            b = 0.4;
            w = 100;

            equilx1
            equilp1

Out[]=      {{x1 → 0.0379196}}
Out[]=      1.84601
```

Finally we plot jointly the demand and supply curves, obtaining the graph shown in Figure 3.9

```
In[]:=      Plot[{p1d, p1s},
            {x1,0.01,0.1},
            AxesLabel → {"x1", "p1"},
            PlotLabel → "Market for x1"]
```

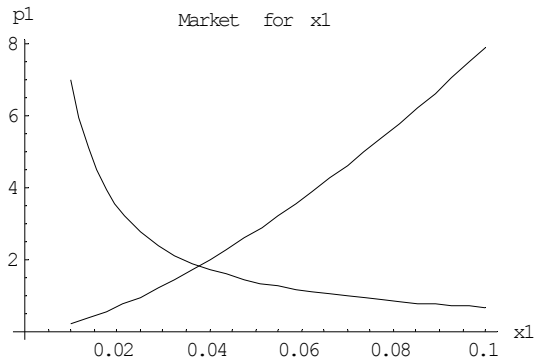


Figure 3.9 Market for x1

Once we obtained the graphical representation of market equilibrium, it is interesting to perform some comparative static exercises. To do so, we use a statement of the form

```
Plot[Evaluate[Table[ ] ] ]
```

This statement nests three Mathematica functions. The function

```
Table[expr, {i, imin, imax, di}]
```

makes a list of the values of an expression `expr` with `i` running from `imin` to `imax` in steps of `di`.. The function

```
Evaluate[expr]
```

causes the expression `expr` to be evaluated. Finally the function `Plot[ ]` is the one we have used before. Thus, the statement below

```
In[]:=      Plot[Evaluate[Table[{p1d ,p1s },{T,1,1.2,0.1}]],
            {x1,0.01,0.1},
            AxesLabel → {"x1", "p1"},
            PlotLabel → "Market for x1"]
```

will first generate a list of three elements, one corresponding to each value of the technology parameter `T`, then evaluate the expression in each element of the list, and finally generate the plot shown in Figure 3.10.

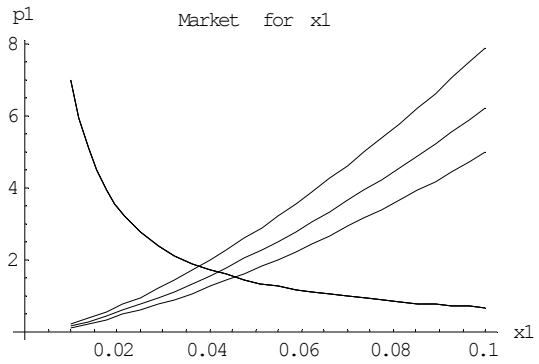


Figure 3.10 Comparative Statics Changing Parameter T

Figure 3.11 shows the result of a similar experiment, but changing the demand function share parameter ρ in the following way

```
{ρ,0.5,0.9,0.2}
```

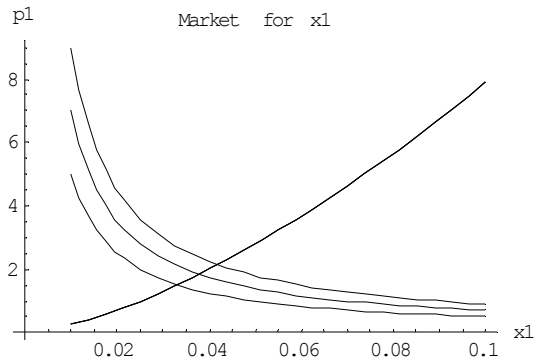


Figure 3.11 Comparative Static Changing Parameter ρ

Finally, we perform the same comparative static exercise now with an animated plot using the following statement

```
In[]:=    Table[Plot[{p1d ,p1s },
          {x1,0.01,0.1},
           PlotRange → {0,8},
           AxesLabel → {"x1", "p1"},
           PlotLabel → "Market for x1"],{T,1,1.2,0.1}]
```

Notice that here we have a `Plot[ ]` function nested within a `Table[ ]` function. Thus, the table will contain a sequence of plots controlled by the evolution of the `T` parameter. The output of the statement will be such a sequence. Double click on the first graph of the sequence and you will see the resulting animation. You can control the speed of the animation with the buttons that will appear at the bottom of the notebook.

Notice that here we fixed the range for the vertical axis with the option `PlotRange`. Otherwise, each plot may generate variable values for that range, creating the false impression that the demand curve is shifting also (to see this, eliminate that

option from the statement and see what happens). Also notice that if you perform other comparative static exercises changing any of the parameters other than `T`, you may have to adjust the `PlotRange` option accordingly as well as the range for `x1`, setting different minimum and/or maximum values.

4. Experiments

A simple set of experiments would be to perform more comparative static exercises changing some model parameters. You may also want to add parameters to the model (e.g. taxes) and see how this affects the outcome of the comparative statics.

Another popular function used to represent preferences or technology is the Constant Elasticity of Substitution (CES) function

$$f(x_1, x_2) = (x_1^{\alpha} + x_2^{\alpha})^{\frac{1}{\alpha}}$$

As we did we the Leontief and Cobb-Douglas functions, you may want to generate the contour plot of this function and see what happens as the parameter α goes from a value near zero to one near minus infinity.

Finally, you may want to develop an analysis analogous to the one we did in this chapter substituting the CES function for the Cobb-Douglas function.

5. Further Readings

For an introduction to Mathematica see Wolfram (2003). Consumer theory and the theory of the firm as well as competitive market equilibrium are at the core of most microeconomics textbooks. Later in this book we will deal with duopoly models in Mathematica and general equilibrium models in GAMS.

Chapter 4

Transportation in GAMS

The transportation problem was made famous among economists by the work of Tjalling Koopmans (1951) and of Robert Dorfman, Paul Samuelson and Robert Solow (1958) a number of whom won the Noble Prize in economics. This kind of model is a most natural way to pose the problem of finding the most efficient place and manner of producing goods and shipping them to customers. The model posits supplies of a good at a number of plants and demands for that good at a number of markets and seeks to find the amount that each plant should ship to each market in order to minimize the transportation cost. Also, the transportation model is the foundation for much more elaborate linear programming industrial models such as those for steel, oil, aluminum, fertilizer and computers. These models focus not only on transportation but also on production and investment.

We begin with a mathematical representation of the transportation problem and then move to a discussion of how this model can be represented in the GAMS software.

1. Mathematical Representation

As an example (adapted from Dantzig (1963)) for this chapter we use the fishing industry with canneries in Seattle and San Diego and markets in New York, Chicago and Topeka (Kansas). In this model we seek to find the pattern of shipments from the canneries to markets which will have the least transportation cost while satisfying the fixed demand at the markets without shipping more from any cannery than its' capacity.

The model is stated mathematically as:

For the sets

I plants = {Seattle, San Diego}

J markets = {New York, Chicago, Topeka}

find the

x_{ij} shipments from plant i to market j

to minimize the transportation cost

(1) $$z = \sum_{i \in I} \sum_{j \in J} c_{ij} \, x_{ij}$$

where

c_{ij} transportation cost from plant i to market j per unit shipped

The criterion function (1) is minimized subject to the constraints that no more be shipped from each plant than its capacity

(2) $$\sum_{j \in J} x_{ij} \le a_i \qquad i \in I$$

where

a_i the capacity of plant i

and that no less be shipped to each market than its demand

(3) $$\sum_{i \in I} x_{ij} \ge b_j \quad j \in J$$

where

b_j the demand at market j

while requiring that all the shipments be non-negative.

(4) $$x_{ij} \ge 0 \quad i \in I \quad j \in J$$

Next we turn to the representation of this model in GAMS.

2. GAMS Representation

GAMS (General Algebraic Modeling System) was developed at the World Bank by Alexander Meeraus and his colleagues. The user's guide for this system is by Brooke, Kendrick, Meeraus and Raman (1998). GAMS was designed as a "set driven" high-level

language that would facilitate the development of linear and nonlinear programming models of industry, agriculture and finance. Thus it was not necessary to write a separate equation for each commodity, time period, crop or equity but rather only to create equations and variables indexed over sets of commodities, time periods, crops, equities etc. In this way a model with thousands of equations could be represented in a GAMS statement with only a few set specifications, variables and equations - all of which might fit on a single page. This not only decreased the tedious, labor-intensive part of model development but also substantially reduced the likelihood of errors in the model specification.

Also, GAMS has become widely used because of the ability to represent in it any model that can be expressed in algebra. In particular there are now many computable general equilibrium, agricultural and financial models in GAMS as well as a wide variety of other types of economic models. For a listing of several hundred GAMS models see the GAMS library that comes with the software or access the library at

http://www.gams.com

These models can be downloaded and solved with the GAMS software.

Many readers of this book will be running GAMS on their home computers or in a computer laboratory in a university. The instructions for fetching the input file and running the program on a personal computer are contained in Appendix A at the end of the book.

The GAMS program corresponding to the transportation problem is available at the book web site under the name `trnsport.gms` as well as in the GAMS library under that same name. (Notice that "transport" is misspelled in this filename.) Also, an extended tutorial on this model is available in the GAMS User's Guide, i.e. Brooke, Kendrick, Meeraus and Raman (1998).

The GAMS language uses a syntax that is reasonably close to mathematics. For example the criterion function for the transportation model is written in mathematics as

$$z = \sum_{i \in I} \sum_{j \in J} c_{ij} \, x_{ij} \tag{5}$$

and in GAMS as

```
cost..    z =e= sum((i,j), c(i,j) * x(i,j)) ;
```

In mathematics equations are usually numbered while in GAMS they are named, thus equation (5) gets the name `cost` and the two dots after `cost` tell GAMS that the

name has been completed and the equation is about to begin. Also, GAMS has an unusual way of representing an equal sign, namely `=e=`. The reason for this is that GAMS also includes less-than-or-equal signs and greater-than-or-equal signs that will be discussed below.

The GAMS language does not make a distinction between set names such as `I` and the indices of the elements which belong to the sets such as `i`. This results partly from the fact that GAMS, unlike most programming languages, does not make a distinction between upper and lower case letters. Thus one could imagine writing the right hand side of the equation above in GAMS as

```
sum((I,J), c(i,j) * x(i,j)) ;
```

to indicate that the sum is over the sets `I` and `J` while the parameter `c` and the variable `x` are defined with the subscripts `(i,j)`. However, that is not necessary in GAMS and the user learns to read symbols like `(i,j)` in GAMS as sometimes representing set names and sometimes representing indices.

Finally, in mathematics the simple juxtaposition of two symbols like c and x indicates that they are multiplied times one another while in GAMS, as in most other programming languages, it is necessary to explicitly indicate multiplication with the asterisk, i.e. `*`.

Similarly, the capacity constraint is written in mathematics as

$$\sum_{j \in J} x_{ij} \le a_i \quad i \in I \tag{6}$$

and in GAMS as

```
supply(i) ..  sum (j, x(i,j)) =l=  a(i) ;
```

So here equation (6) gets the name `supply` and the `(i)` that follows it indicates that there is one constraint of this type for each element in the set I, i.e. for each plant. Thus the `(i)` next to the equation name in GAMS plays the same role as the symbols $i \in I$ play in the mathematics. Also notice that the less-than-or-equal-to sign, $\le$ in mathematics becomes `=l=` where the `l` here indicates the letter `l` and not the number `1`.

The transportation model as stated above could be for a model with two plants and three markets or for a model with 50 plants and 200 markets since we have so far not specified the sets I and J nor the parameters a, b and c. This is one of the powers of

the GAMS language, i.e. one can write a model prototype which can be used for any of a number of industries and then specialize it in the set specifications and the parameter definitions to a particular industry in a chosen country.

So consider next how the sets are specified in the GAMS language. In this model there are two plants and three markets. The set of plants is specified in mathematics as

$$I = \{Seattle,\ San-Diego\}$$

The equivalent GAMS statement is

```
i = / seattle, san-diego /
```

GAMS uses forward slashes as set delimiters while mathematics use braces.

Once the sets are specified then the data can be input using the `parameter` and `table` keywords as shown below. Consider first the use of the `parameter` keyword to input the capacity and demand data.

```
Parameter
      a(i)  capacity of plant i in cases
        /    seattle     350
             san-diego   600  /

      b(j)  demand at market j in cases
        /    new-york    325
             chicago     300
             topeka      275  / ;
```

Observe that the parameter "`a`" is followed by the set over which it is defined, i.e. it is written as "`a(i)`". As was mentioned earlier in the book, it is not necessary to include the set here; however it is a useful precaution because when the set is provided the GAMS complier can check to be sure that all the element names used in the input of the parameter do indeed belong to that set.

Next consider the input of the distance data with the statement

```
Table d(i,j)  distance in thousands of miles
                  new-york       chicago      topeka
    seattle          2.5           1.7          1.8
    san-diego        2.5           1.8          1.4  ;
```

Here the `table` keyword is used to input the matrix of transportation distances between the markets and the plants.

Next consider the `scalar` keyword that can be used to input a scalar quantity, in this case, `f`, which is freight cost per case per thousand miles.

```
Scalar f  freight in dollars per case per thousand miles  /90/ ;
```

This scalar in turn can be used in a parameter statement to compute the transport cost per case between each plant and market as shown below.

```
Parameter c(i,j) transport cost in thousands of dollars per case ;
              c(i,j) = f * d(i,j) / 1000 ;
```

Notice here that the new element `c(i,j)` is first declared with a parameter statement and then defined with a mathematical statement in which `f` is multiplied by `d` and divided by 1000. Here you see that the `parameter` keyword in GAMS is much more versatile than just being used to input vectors.

The computation of the `c(i,j)` parameter above illustrates one theme in the use of the GAMS language. The user is encouraged to enter the raw data for the model in the GAMS statement and to show explicitly all the mathematical transformations which are performed on that data before it become a part of the model equations.

Consider a word of warning about the data for the transportation model from the GAMS library. Notice that the distances in `Table d(i,j)` above are listed as the same from Seattle to New York and from San Diego to New York, i.e. 2,500 miles. This can cause multiple optimal solutions to the model and this can be confusing. Therefore, when first using this model it is probably wise to make these distances different. For example one might change the number for the Seattle to New York distance from 2.5 (thousand miles) to 2.7 (thousand miles).

Next consider the variables and equations part of the GAMS representation that is shown below.

```
Variables
     x(i,j)  shipment quantities in cases
     z       total transportation costs in thousands of dollars ;

Positive Variable x ;
Equations
     cost        define objective function
     supply(i)   observe supply limit at plant i
     demand(j)   satisfy demand at market j ;

cost ..        z  =e=  sum((i,j), c(i,j)*x(i,j)) ;

supply(i) ..   sum(j, x(i,j))  =l=  a(i) ;

demand(j) ..   sum(i, x(i,j))  =g=  b(j) ;
```

The keyword `Variables` is used to declare the variables and in the process one indicates the sets over which the variables are defined. For example the variable x is defined for the set of plants I and the set of markets J so it is listed as `x(i,j)`. The restriction that the shipment variables must be non-negative as shown in Equation (4) above is carried in the `Positive Variable x` statement.

The names of the equations are listed after the `Equations` keyword along with the sets over which they are defined. For example there is a supply equation for each plant so that equation is defined as `supply(i)`.

The final statements in the GAMS specification are listed below.

```
Model transport /all/ ;
Solve transport using lp minimizing z ;
Display x.l, x.m ;
```

The `Model` keyword is used to give the model a name - in this case `transport` - and to indicate the equations which are included in the model. One may either list a subset of the equation names here or if the model consists of all the equations listed above then the `all` keyword can be used. The model is then solved with the `Solve`, `using` and `minimizing` keywords. From a mathematical point of view, the transportation problem is a particular case of what is known as "linear programming", that is, a problem in which one seeks to optimize a linear objective function subject to a set of linear constraints. For an introduction to linear programming see Appendix G. The `lp` in the solve statement tells GAMS to use its linear programming solver to compute the solution to the model and the `z` is the criterion value that is to be minimized. Since the model contains indexed equations, GAMS will use a stacking method as discussed in Appendix H.

Finally, the `display` statement requests that the activity levels for the shipment variables, i.e. `x.l` and the marginal values `x.m` for these same variables be displayed in tables.

Learning all this syntax for a programming language may at first seem complicated. However, the structure of the model helps to simplify things. Notice in the complete GAMS statement of the model which follows this paragraph that the model is defined in steps

first the sets
then the parameters
then the variables
then the equations
and finally the model and solve statements.

It takes a while to adjust to all the details but the overall structure and form of a GAMS representation of a model can be grasped quickly.

The entire GAMS listing of the model is presented below.

```
$Title  A Transportation Problem (TRNSPORT,SEQ=1)

  Sets
       i   canning plants   / seattle, san-diego /
       j   markets          / new-york, chicago, topeka / ;

  Parameters
       a(i)  capacity of plant i in cases
         /    seattle     350
              san-diego   600  /
       b(j)  demand at market j in cases
         /    new-york    325
              chicago     300
              topeka      275  / ;

  Table d(i,j)  distance in thousands of miles
                    new-york       chicago      topeka
      seattle          2.5           1.7          1.8
      san-diego        2.5           1.8          1.4  ;

  Scalar f  freight in dollars per case per thousand miles  /90/ ;

  Parameter c(i,j)  transport cost in thousands of dollars per case ;

            c(i,j) = f * d(i,j) / 1000 ;

  Variables
       x(i,j)  shipment quantities in cases
       z       total transportation costs in thousands of dollars ;
  Positive Variable x ;

  Equations
       cost        define objective function
       supply(i)   observe supply limit at plant i
       demand(j)   satisfy demand at market j ;

  cost ..        z  =e=  sum((i,j), c(i,j)*x(i,j)) ;

  supply(i) ..   sum(j, x(i,j))  =l=  a(i) ;

  demand(j) ..   sum(i, x(i,j))  =g=  b(j) ;

  Model transport /all/ ;

  Solve transport using lp minimizing z ;
  Display x.l, x.m ;
```

This completes the discussion of the input for the model. Next we turn to the way to solve the model and a discussion of the results.

3. Results

As was discussed above, Appendix A contains instructions on how to access the *.gms file from the GAMS library, how to solve the model and how to examine the results by using the listing file, *.lst. This last step can seem complicated at first because the GAMS output files contain a substantial amount of information about the structure of the model and its solution. However, it is simple enough to jump around in the file to examine the key parts.

One should first locate the Solve Summary part of the output. To do this search in the editor for the string "SOLVER STATUS". When you do so you will see a section of the output that looks like

```
      S O L V E      S U M M A R Y

     MODEL   TRANSPORT           OBJECTIVE  Z
     TYPE    LP                  DIRECTION  MINIMIZE
     SOLVER  BDMLP               FROM LINE  70

**** SOLVER STATUS     1 NORMAL COMPLETION
**** MODEL STATUS      1 OPTIMAL
**** OBJECTIVE VALUE               153.6750
```

Each time after you solve a GAMS model you should check this section of the output to be sure that the model was solved successfully. The words NORMAL COMPLETION here indicate that is the case. If the solution procedure was not successful you will find words like INFEASIBLE or UNBOUNDED. Be on guard against the fact that the GAMS output will provide a solution to the model even when that solution is infeasible. However, the solution provided would not be the optimal solution but rather the last one tried before it was determined that the solution was infeasible. For this reason it is particularly important to check the SOLVER STATUS and MODEL STATUS after each run and before the solution variables are used.

Next skip down the output across the sections labeled "---- EQU" until you get to the sections labeled "---- VAR" which looks like

```
---- VAR X           shipment quantities in cases

                      LOWER     LEVEL     UPPER    MARGINAL

SEATTLE  .NEW-YORK      .       50.000     +INF       .
SEATTLE  .CHICAGO       .      300.000     +INF       .
SEATTLE  .TOPEKA        .         .        +INF      0.036
SAN-DIEGO.NEW-YORK      .      275.000     +INF       .
SAN-DIEGO.CHICAGO       .         .        +INF      0.009
SAN-DIEGO.TOPEKA        .      275.000     +INF       .
```

The interesting part here is the activity level of the shipment variables x in the column labeled LEVEL. This shows, among other things, that 50 cases were shipped from Seattle to New York and 300 cases were shipped from Seattle to Chicago. This is the solution of the model that we were looking for. These same results are shown a little further down in the output in a section labeled VARIABLE X.L which is the result of the display statement in the GAMS input. That output is shown below.

```
----     72 VARIABLE  X.L           shipment quantities in cases

             NEW-YORK     CHICAGO      TOPEKA

SEATTLE        50.000     300.000
SAN-DIEGO     275.000                 275.000
```

This table is somewhat easier to read than the default output and thus you can see the reason that most GAMS input files end with a series of display statements. These tables are easily found since they are at the end of the long GAMS output so the user can quickly scroll to the bottom of the file and find the key results. However, they will be there only if you remember to add a display statement at the end of the GAMS input statement.

There is just one other key piece of the GAMS output file which we should look at before we turn our attention elsewhere. It is in the "---- EQU" section that we skipped over earlier and that you can quickly find by scrolling back up to it or by searching for it with the editor. The part of interest is the equation-wise output for the demand constraints that looks like

```
---- EQU DEMAND      satisfy demand at market j

             LOWER     LEVEL     UPPER    MARGINAL

NEW-YORK   325.000   325.000     +INF       0.225
CHICAGO    300.000   300.000     +INF       0.153
TOPEKA     275.000   275.000     +INF       0.126
```

In this case we are interested in the MARGINAL column. These values are called "shadow prices" or "dual" variables and have important economic meaning. They show us that for each additional unit of demand at New York the objective function will have to increase by .225 but by only .153 at Chicago and .126 at Topeka. These numbers are like prices and indicate that it is substantially more expensive to supply fish to New York than to Chicago or Topeka. Similar numbers in electric power models can be used by regulators to determine the price of power in cities that are nearby or far away from electric power generation facilities such as dams, nuclear plants or coal burning plants.

So in summary, when looking at the GAMS output you should first check to be sure that the problem was solved satisfactorily. Then focus on the variables section and finally take a look at the equation section.

4. Experiments

As a simple experiment, one might first change the number for the Seattle to New York distance from 2.5 (thousand miles) to 2.7 (thousand miles) - to eliminate the multiple solution problem discussed above - and then solve the model again. Or one might decrease the demand at one or more markets or increase the supply at one or more plants in order to analyze the effects on the optimal solution. However, when changing the supply and demand parameters one must be careful to assure that the total supply is greater than or equal to the total demand – otherwise the solution to the model will be infeasible.

A more complicated experiment is to add additional markets and/or plants. This helps one to learn quickly how the sets are specified and the ripple effect this has on required changes in the `parameter` and `table` statements. In the process one may switch the model from a focus on fish to steel or fertilizer or glass or computers or whatever industry is of interest.

Yet more complicated would be to add production cost at each plant. This could be done by introducing a new parameter as follows:

```
Parameters
     prodcost(i)  production cost of plant i per case
       /    seattle     15
            san-diego   18  / ;
```

Then the criterion function would also need to be changed from

```
cost ..     z  =e=  sum((i,j), c(i,j)*x(i,j)) ;
```

to

```
cost ..     z  =e=  sum((i,j), (c(i,j) + prodcost(i))*x(i,j) );
```

Then the model can be used to analyze the effects of production differences at plants as well as transportation cost differences between pairs of plants and markets.

If it is desirable to change the criterion from cost minimization to profit maximization this can be done by introducing prices at each market. One way to do this is by approximating a nonlinear demand function with a piecewise linear function. For example the demand might be thought of in three segments with the total demand in market j being equal to the sales in the three segments, viz

$$s = s_1 + s_2 + s_3$$

The subscript j for the market is omitted here in order to simplify the notation in the following development.

The revenue generated by sales at this one market could then be written

$$rev = p_1 s_1 + p_2 s_2 + p_3 s_3$$

while being careful to insure that the parameters for the price in the first segment is higher than the price in the second segment that is, in turn, higher than the price in the third segment, i.e.

$$p_1 > p_2 > p_3$$

and putting an upper bound on sales in each of the first two segments, i.e.

$$s_1 \le \beta_1$$
$$s_2 \le \beta_2$$

where β_1 is the upper bound in segment one and β_2 is the upper bound in segment 2.

Then the criterion value becomes the maximization of profit which is revenue minus cost, that is

$$\pi = \text{rev} - \text{cost}$$

where

$$rev = \sum_{j \varepsilon J} \left(p_{1j} s_{1j} + p_{2j} s_{2j} + p_{3j} s_{3j} \right)$$

where

p_{kj} = price in the kth segment in market j
s_{kj} = sales in the kth segment in market j

and the cost includes the transportation and production cost.

5. Further Reading

In the 1970's and 80's there was a project at the World Bank under the leadership of Hollis Chenery and more directly of Ardy Stoutjesdijk which focused on the development of a variety of industrial models for steel, fertilizer, pulp and paper and other industries. These models used the GAMS language which was under development at that time at the World Bank by Alexander Meeraus and his colleagues. One of those models, namely the one on the Mexican steel industry, is a logical follow-on to the model developed in this chapter, viz Kendrick, Meeraus and Alatorre (1984). Also, there is a shorter, more intuitive, chapter on this model in Kendrick (1990) and various versions of the model itself are available in the GAMS library.

Chapter 5

Databases in Access

Database systems have had very substantial impact on the way that both businesses and government agencies manage production, sales, inventory and personnel. Curiously though, they have as yet had relatively little impact on the way economists develop and maintain the data which are used to measure the pulse of economic activity in both macro and micro economic settings. It seems likely that this will change as a new generation of economists who have cut their teeth on Mac's and PC's arrive on the scene.

This chapter provides an introduction to the use of relational database systems using the Access software. An example database developed by Kendrick (1982b) in Access is used to illustrate the potential for relational database systems in economics.

At present most economic data is organized as sets of unrelated time series which are maintained by different agencies. Thus to find the consumption data for the U.S. economy one might go Bill Goffe's "Resources for Economists" page on the Internet (see Goffe (2004)) and from there track down the macroeconomic databases and pull out the consumption time series. From an econometricians point of view this may be a very serviceable system. Thus to estimate a consumption function the user might download the time series for consumption, income, taxes and interest rates into a spreadsheet such as Excel or an estimation package. Then disposable income would be calculated from the income and taxes series. Finally consumption would be regressed on disposable income and interest rates.

However, there are many other uses of economic data than as inputs to regression packages. Frequently the user does want to run a regression but rather to address a query that depends upon the relationships between the data. If the data is stored and organized, not as a set of unrelated time series, but rather as a relational database, such queries can be answered quickly and easily. Moreover, once the data were organized in relational forms this would also serve the econometricians very well by permitting easy

control over aggregation and disaggregation and development of samples for use in estimation packages.

This chapter begins with an introduction to the terminology of database systems. The example database for the U.S. economy is outlined along with the specification of this data in the Access software. Then the procedure for developing and using queries of the database is discussed.

1. Domains, Relationships and Joins

The relationship is the key concept in database methodology. Yet it is as simple as a table. For example, consider a table which shows the locations and production levels of a set of plants as in Table 5.1.

Plant	**City**	**Commodity**	**Output**
Inland	Gary	Steel	4
ARCO	Houston	Oil	73
Alcoa	Rockdale	Aluminum	125

Table 5.1 Production Relationship

Thus the Inland Steel plant at Gary, Indiana produced 4 million tons of steel, the ARCO refinery at Houston, Texas processed 73 thousand barrel of oil and the Alcoa aluminum smelter at Rockdale, Texas produced 125 thousand tons of aluminum. In the language of database systems this table would be called a relationship of the form

Production (Plant, City, Commodity, Output)

and the domains of the relationship would be the sets of plants, cities, commodities and output levels used in the database.

The Production relationship above would have three elements and each element would be a four-tuple, i.e.

Production = { (Inland, Gary, Steel, 4)
(ARCO, Houston, Oil, 73)
(Alcoa, Rockdale, Aluminum, 125) }

So, in summary, the Production relationship is a set with elements, each of which is a tuple.

Another key concept in relationship databases is that of the "join". In order to illustrate a join we introduce two more relationships, City-State and State-Region as shown in Tables 5.2 and 5.3.

City	**State**
Gary	Indiana
Houston	Texas
Rockdale	Texas

Table 5.2 City-State Relationship

State	**Region**
Indiana	Mid-West
Texas	Gulf-Coast

Table 5.3 State-Region Relationship

The City-State and State-Region relationships have a common "domain", i.e. State. So one can join these two relationships to create a new relationship which shows the region of each city, i.e.

City	**Region**
Gary	Mid-West
Houston	Gulf-Coast
Rockdale	Gulf-Coast

Table 5.4 City-Region Relationship

Furthermore, one can then do an additional join of the Production and the City-Region relationships using the common "City" domain to obtain a Regional Production relationship which is shown in Table 5.5.

Region	Commodity	Plant	Output
Mid-West	Steel	Inland	4
Gulf-Coast	Oil	ARCO	73
Gulf-Coast	Aluminum	Alcoa	125

Table 5.5 Regional Production Relationship

This table could then be printed without the Plant domain to produce the desired result as shown in Table 5.6.

Region	Commodity	Output
Mid-West	Steel	4
Gulf-Coast	Oil	73
Gulf-Coast	Aluminum	125

Table 5.6 Regional Production

All of this may seem like a lot of work to obtain a simple table. However, notice that the State-Region relationship can be modified independently of the others. Thus an economist would be free to create his or her own regional aggregation scheme and develop queries based on that scheme.

2. An Example Database

An example database for the U.S. economy from Kendrick (1982b) is provided in full in Appendix 5A at the end of this chapter. The Access file is available on the book web site. That database was created as a simple illustration of how a relational database might be constructed with data from the U.S. economy. The purpose was not be comprehensive but rather to illustrate how one might fruitfully link together production, ownership, labor relations, ownership, location and even politics in a single database.

The U.S example database is a set of fourteen relationships which link together commodities, productive units, plants, unions, companies, industries, sectors, cities, states, regions, governors and political parties in the fashion outlined in Figure 5.1.

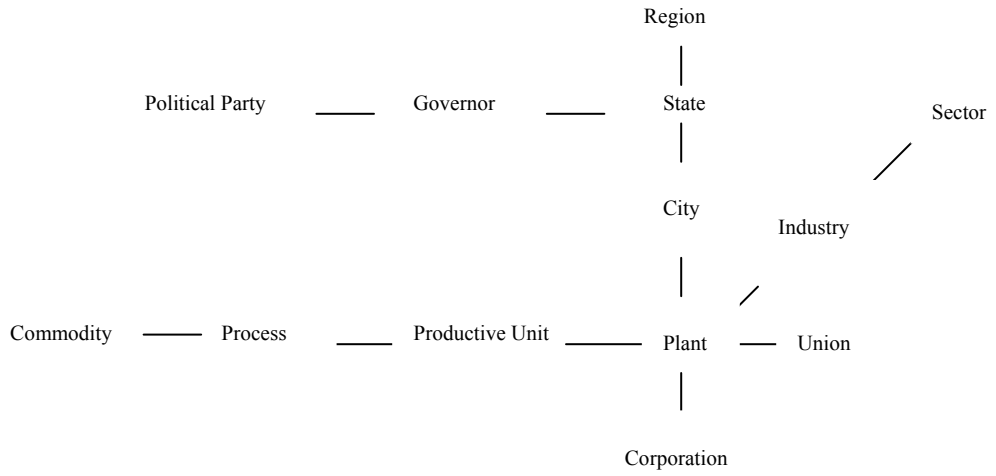


Figure 5.1 Links between the Domains in the Example Database

The story which can be told about how these domains are linked is

plants contain *productive units* in which *processes* are used to produce *commodities*
plants belong to *corporations*
plants have workers who belong to *unions*
plants belong to *industries* which belong to *sectors*
plants are located in *cities* which are in *states* which are in *regions*
states have *governors* who belong to *political parties*

The same story can be told using the relationships instead of the domains. The inputs to and outputs from production processes are described by

Input-Output (commodity, process, input-output coefficient)

and the level of production of each commodity by each process is given by

Production (commodity, process, year, production level).

The productive units which are used by each process are indicated in

Capacity Use (process, productive unit, capacity coefficient)

and the capacity of those productive units in each plant are shown in

Capacity (productive unit, plant, year, capacity level).
The increase in this capacity in a given year at each plant is displayed in
Increment to Capacity (productive unit, plant, year, incremental capacity).

The plants are owned by corporations
Ownership (plant, corporation).
Also the plants have employees who belong to unions
Plant Employees (plant, union, number of employees).
Moreover the plants belong to industries
Industry Composition (plant, industry)
and the industries to sectors in the economy
Sector Composition (industry, sector).

The plants are located in cities
Plant Location (plant, city)
which are located in states
City Location (city, state)
which are in turn located in regions of the country
State Location (state, region).
The states have governors
State Governors (state, governor)
who are affiliated with political parties
Party Affiliation (governor, party).

One would not be able to tell such a simple story for a full and comprehensive database of the U.S. economy, but this simple story and small database serve well our purpose to introduce the use of relational databases in economics.

3. Representation of the Example Database in the Access Software

We turn now to how this relational database is represented and used in the Access software. When you open this database in Access the first window you see will contain a list of the tables (relationships) which makeup the database as is shown in Figure 5.2 below.

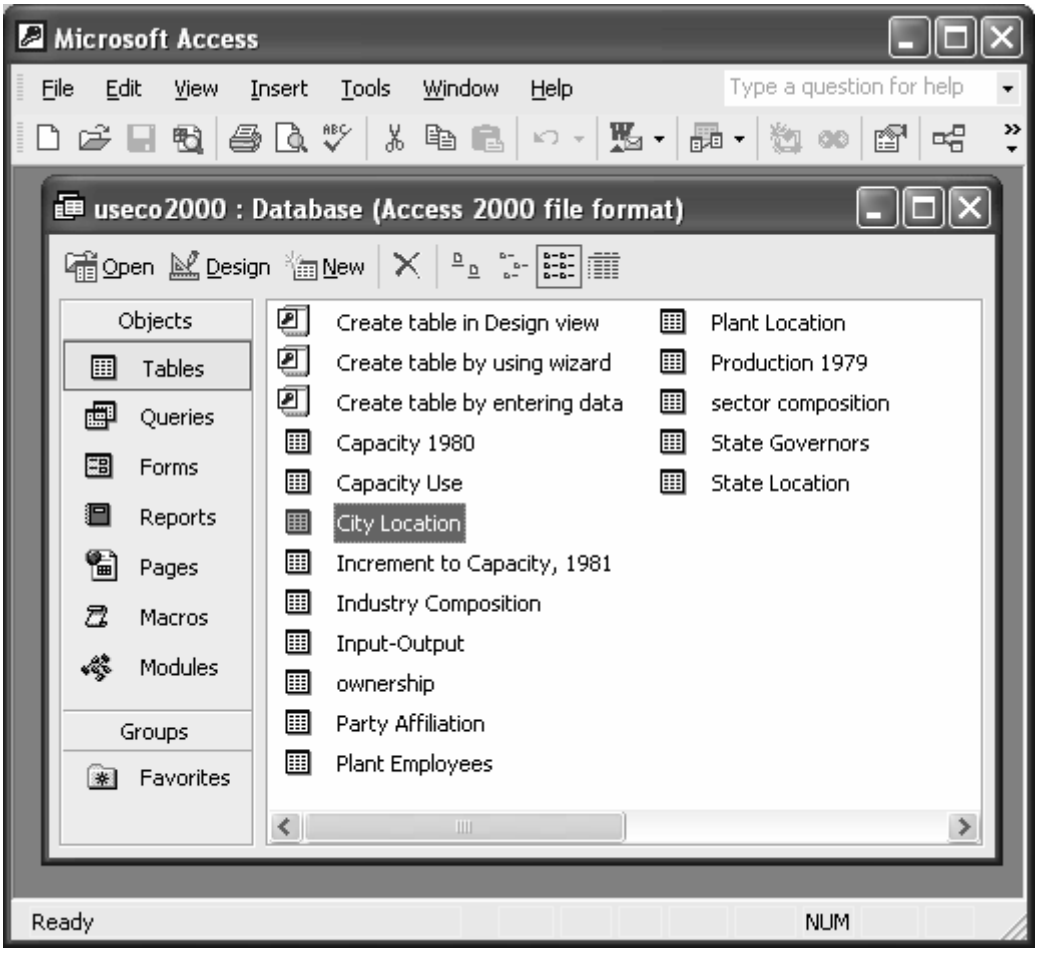


Figure 5.2 Tables in the U.S. Economy Example Database

Figure 5.2 shows the fourteen relationships which were discussed above. Unfortunately they are in alphabetically rather than the conceptual order used above. You can examine the tables one-by-one by double clicking on them and then comparing them to the corresponding relationship in Appendix 5A. In particular take a look at the Ownership and the Plant Employees tables since we will use both of them in the explanation of the query below.

The principal use of databases is to answer queries, i.e. questions about the data in the database. First we will take a look at a couple of existing queries and how they are specified in Access. Select the "Queries" option in the objects bar on the “useco2000 : Database” window and you will see the display shown below in Figure 5.3.

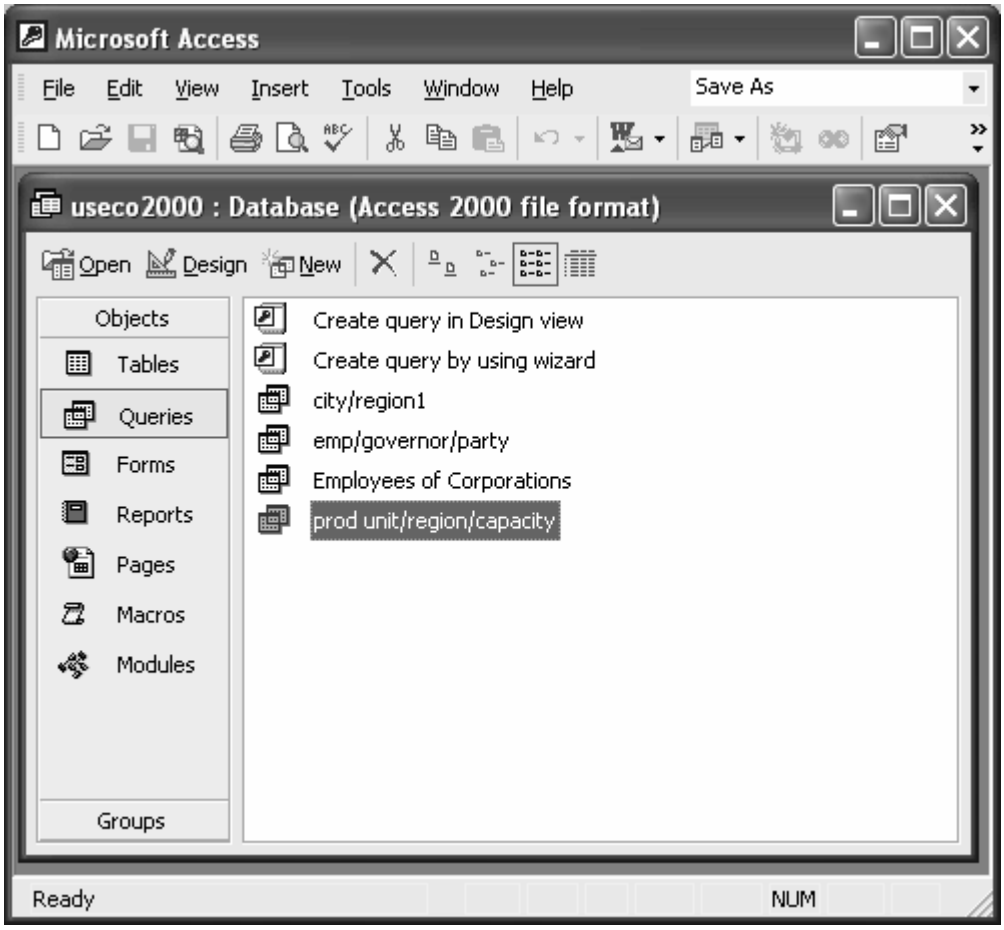


Figure 5.3 Queries

This window shows that four queries have already been developed. After you have seen how they work you will be in a position to develop queries of your own. Consider first the Employees of Corporations query. Select it but be careful to single click on it rather than to double click. This query answers the question of how many employees of each corporation are members of each union. One cannot answer this question directly by looking at the individual tables; however the question can be answered by combining the information in the two relationships

Ownership (plant, corporation)
Plant Employees (plant, union, number of employees)

The first tells us which corporation owns each plant and the second tells us how many employees in each plant belong to each union. Thus if we combine the two we have a new relationship which we call "Employees of Corporations" and that has the domains

Employees of Corporations (union, corporation, number of employees)

This is a "join" of the type we discussed at the beginning of the chapter since we are joining two relationships together by using the common domain “plant”.

Look back to Fig. 5.3 above and notice that at the top of the Queries window there is a Design button in the toolbar. Click that button and a window will open which shows the design of the query as in Figure 5.4.

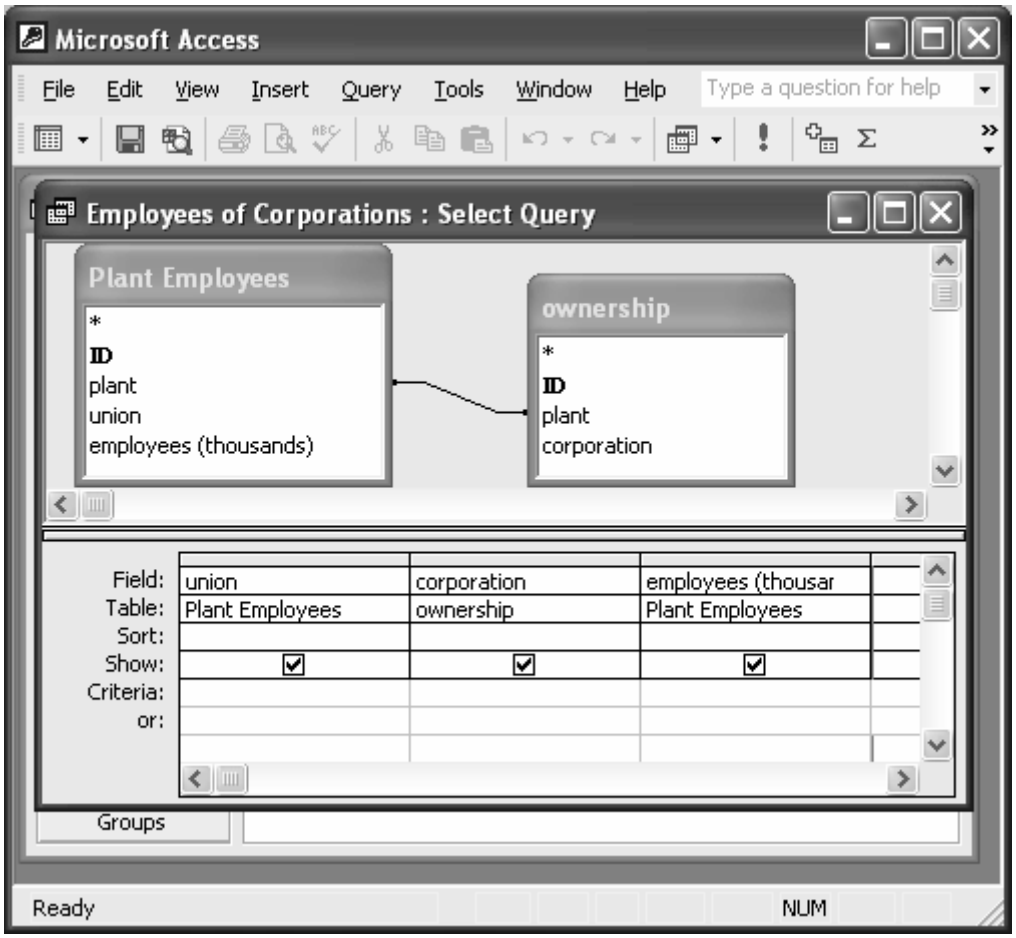


Figure 5.4 Design of the Employees of Corporations Query

The top part of this window includes the two relationships which are used in the query, namely Plant Employees and Ownership. There is a small zigzag line which connects the plant domain in the two relationships. This indicates that the join is to be performed over this domain. You can move the Plant Employees and the Ownership table around by clicking on the label at the top and dragging the table. This is a capability which will come in handy later when you begin designing your own queries.

The bottom half of the window in Fig. 5.4 contains a table in which you see listed the domains that will be in the new relationship that is created by the query. Also there are check marks which allow you to suppress the display of any of the domains. We need all of them for our query so leave all the check marks for the moment and close the query design window by clicking on the "x" in the upper right hand corner.

Now you will be back at the "Query" window. Be sure that the Employees of Corporations query is still selected and then click on the Open button in the toolbar at the top of the dialog box. The window which is shown in Fig. 5.5 will appear.

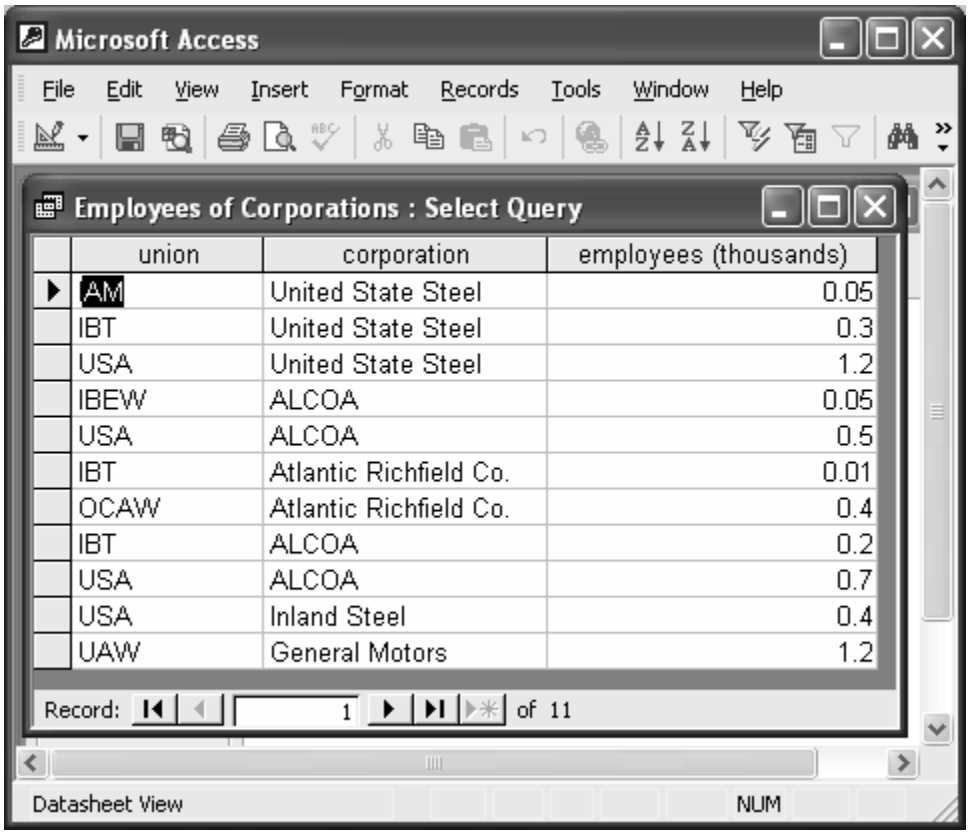


Figure 5.5 Answer to the Employees of Corporations Query

Notice here that the answer to a query is itself a relationship. You can quickly see from the table that U. S. Steel has employees who belong to the machinists (IAM), teamsters (IBT) and steel workers (USA) unions. Notice that there are also two lines in the table which are almost identical

USA	ALCOA	0.5
USA	ALCOA	0.7

This happens because there are two plants in the database which are owned by ALCOA, namely Rockdale and Point Comfort and there are members of the United Steel Workers of America (USA) employed at both plants. From this you can see that it is sometimes necessary to aggregate after a query is run before you have the answer in exactly the form you want.

Next try designing a query of your own. We will use one the queries that is already available in the database so that you can see how it should come out in case yours does not work out as it should. We will take the simplest case. There are relationships for the location of cities by state and for the location of states by region so use these two to create a relationship which show the locations of cities by region. Thus we will use the two relationships

City Location (city, state)
State Location (state, region)

to create a new relationship which we will call

City/Region2 (city, region)

The name "City/Region2" is used to distinguish the query from the one already in the database called "City/Region1".

Here you see the principles of designing your own query. Begin by clicking on the Query object in the objects bar and without selecting any of the existing queries click on the "New" button in the toolbar at the top of the window. When you do this the dialog box shown in Figure 5.6 will appear as shown below.

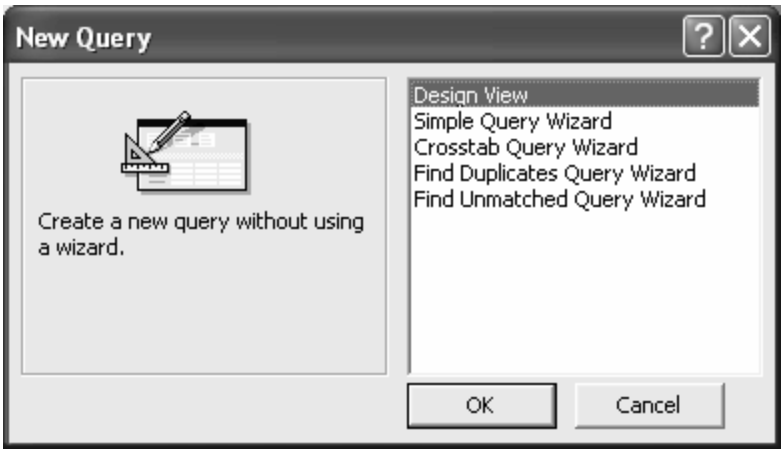


Figure 5.6 New Query Dialog Box

Though you may find it useful to use one of the Wizards later lets do it by hand here. So select the "Design View" option and click on OK. The "Show Table" dialog box will appear as in Figure 5.7.

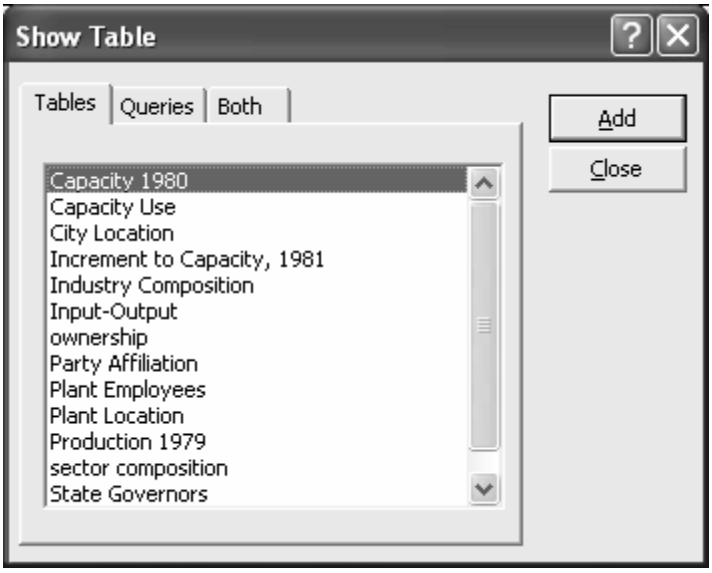


Figure 5.7 Show Table Dialog Box

One of the tables we want to use in the query is City Location, so click on it and then click on the "Add" push button in the upper right hand corner of the dialog box. The City Location table will appear in the top half of the query design dialog box. Next select from the "Show Table" dialog box the State Location table (scroll down to find it if necessary) and then click on the "Add" push button in the upper right hand corner of the

Show Table dialog box. Then close the Show Table dialog box. Once you have done this the window should look something like that shown in Figure 5.8.

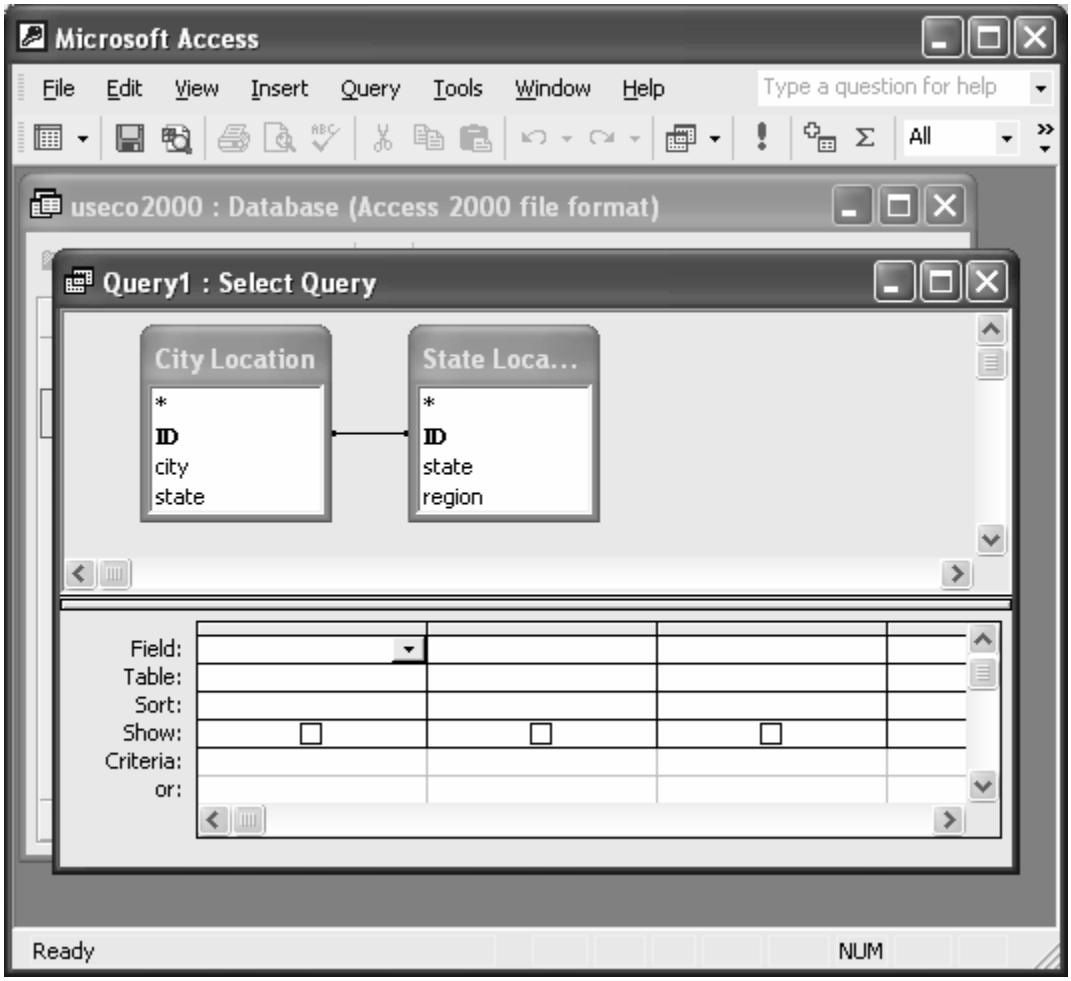


Figure 5.8 Designing a Query

The City Location and State Location relationships should both be displayed in the top half of the query design window as is shown in Figure 5.8.

Before we go further change the name of the query from the default to the choice discussed above of "City/Region2". To do this select the File menu and the Save option. A small dialog box will appear which will allow you to rename the query. Do so and then close that small dialog box and you will be back at Fig. 5.8.

The next step is to establish the join. In Fig. 5.8 there is already a join between the ID's in the two tables but we do not want this. So click on the line which connects the two relationships and strike the "Delete" key so that the line disappears. You may have

trouble with this at first but keep trying until when you click on the line it becomes slightly darker to indicate that it has been selected. Then you should be able to delete it by striking the "Delete" key.

Next create a join between the "state" domain in the two relationships by clicking on "state" in one of the tables and dragging to “state” in the other table. Once you have done this the window should appear as shown in Fig. 5.9.

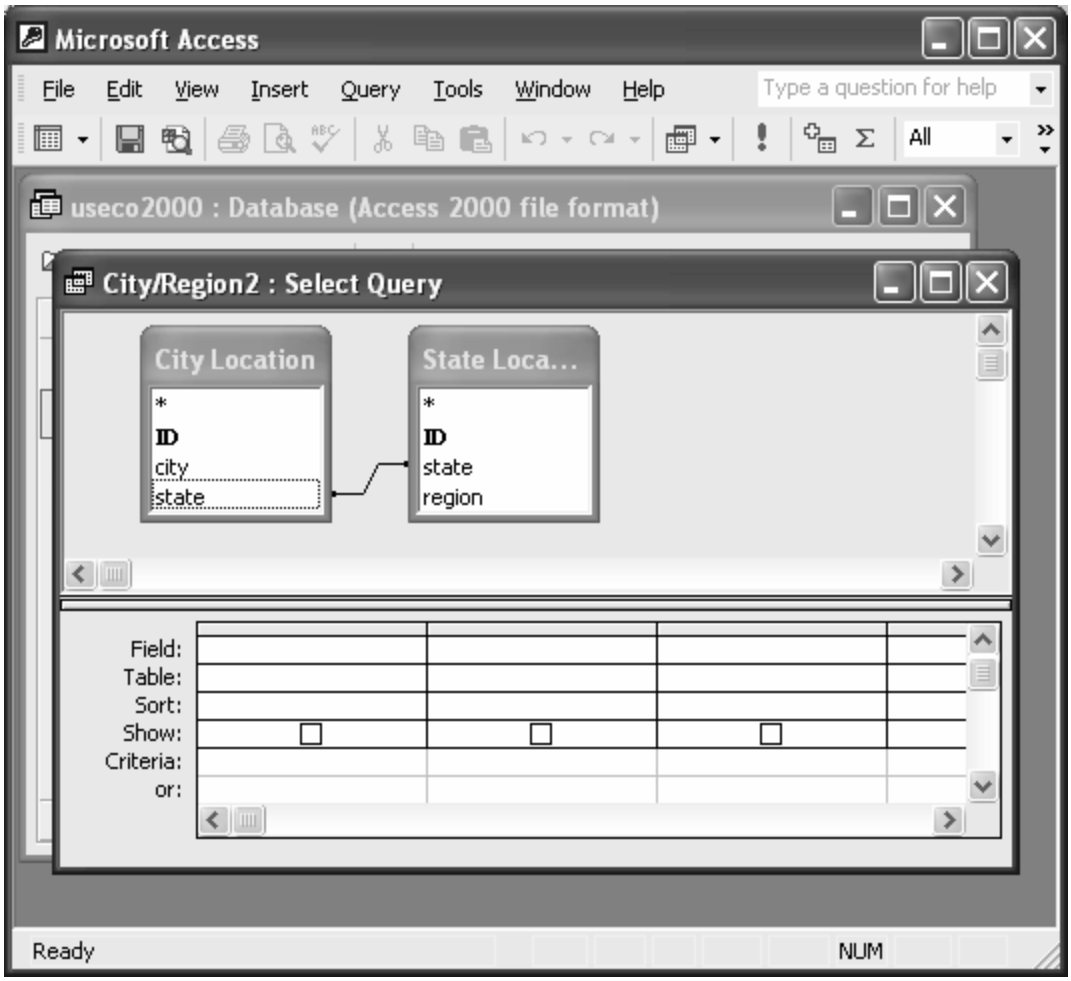


Figure 5.9 A Join Between "State" in the Two Relationships

So the zigzag line between the "state" domains in the two relationships indicates that a join has been established. This completes our work on the top part of the design window and we can now turn our attention to the bottom part.

Begin by clicking in the first column at the "Field" row. When you do so the cursor will appear there along with a small arrow in the right hand side of the box. Click on this box to cause a drop-down window to appear as is shown in Fig. 5.10.

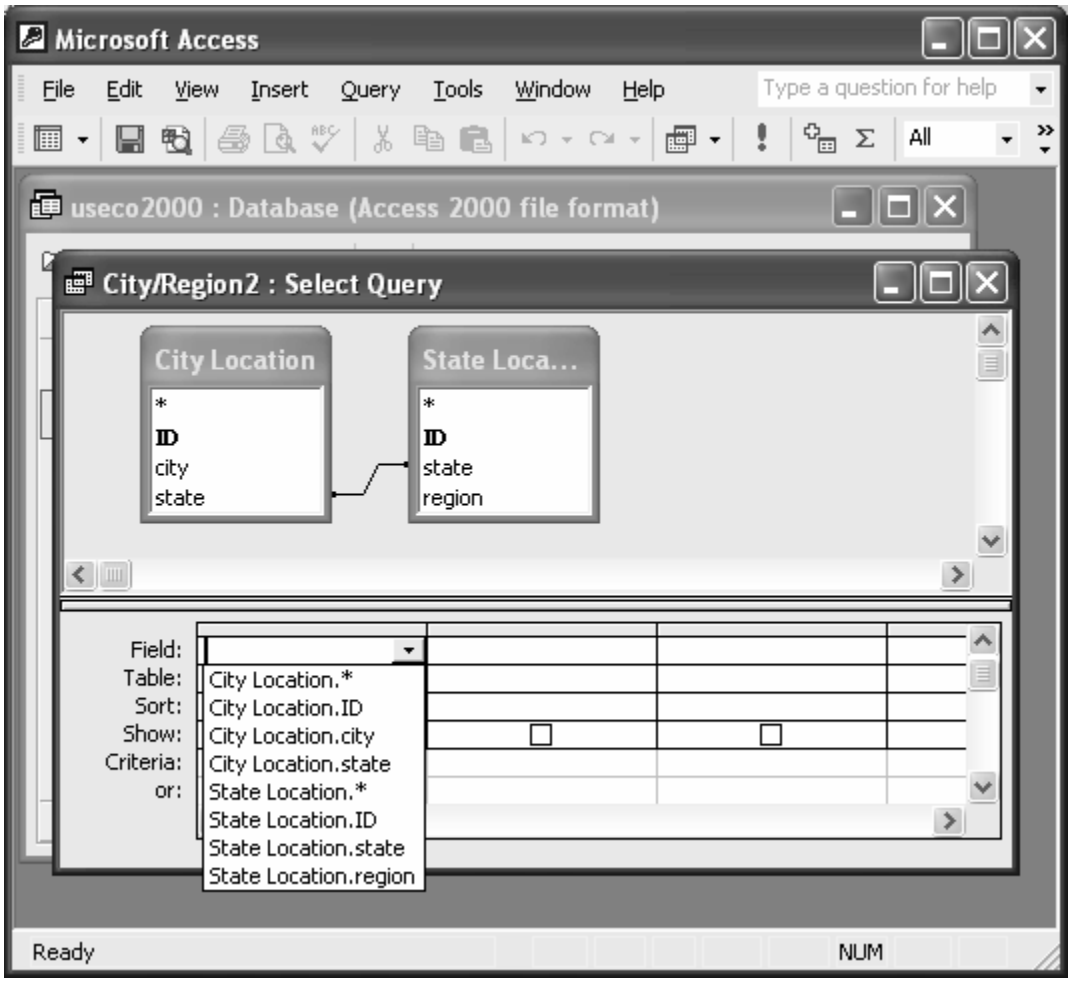


Figure 5.10 Drop-Down Window for Filling in the Field

Since we want the "city" domain in this first field select the line "City Location.city" and the domain "city" will then appear in the box. Repeat this process for the Field row in the next column but this time select "State Location.region" from the drop-down window. Once you have done this the Query Design window should appear as is shown in Figure 5.11.

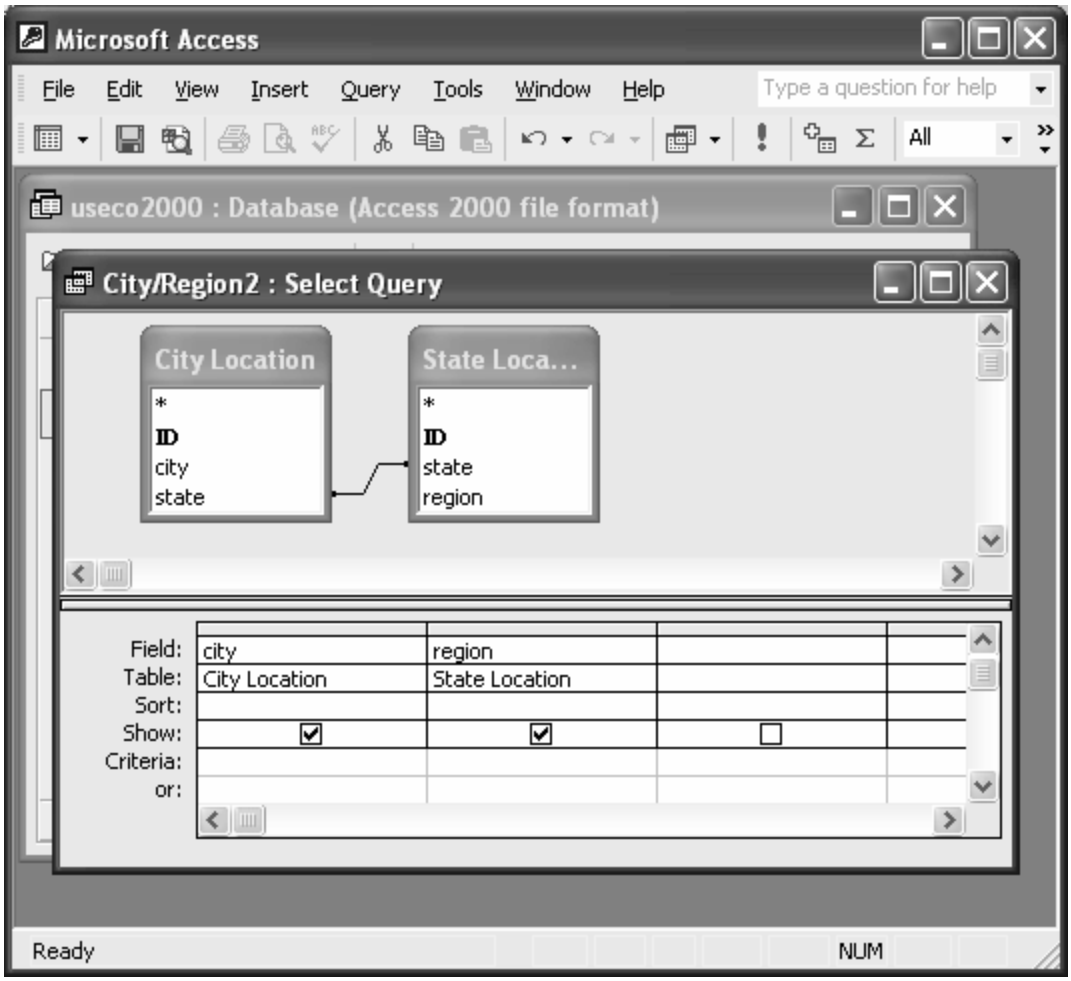


Figure 5.11 Completed Query Design

The query design process is now completed. This undoubtedly seems like a long and complicated process but it goes very quickly once you have the hang of it.

Now we are ready to make use of the query which we have constructed. To do this close the Query Design window by clicking on the "x" in the upper right hand corner. When you do this the Query dialog box should appear as shown in Figure 5.12.

Figure 5.12 Query Dialog Box Showing the New Query "City/Region2"

Now the Query dialog box contains the new query which we have created "City/Region2". As mentioned earlier there was already a "city/region1" query which performs the same function and to which you can compare if you have had difficulties in some of the above steps. However, for now ignore this and try the one we have created by clicking on"City/Region2" and then clicking on the "Open" push box in the upper left hand corner of the window. When you do so the result of the query should appear as shown in Figure 5.13.

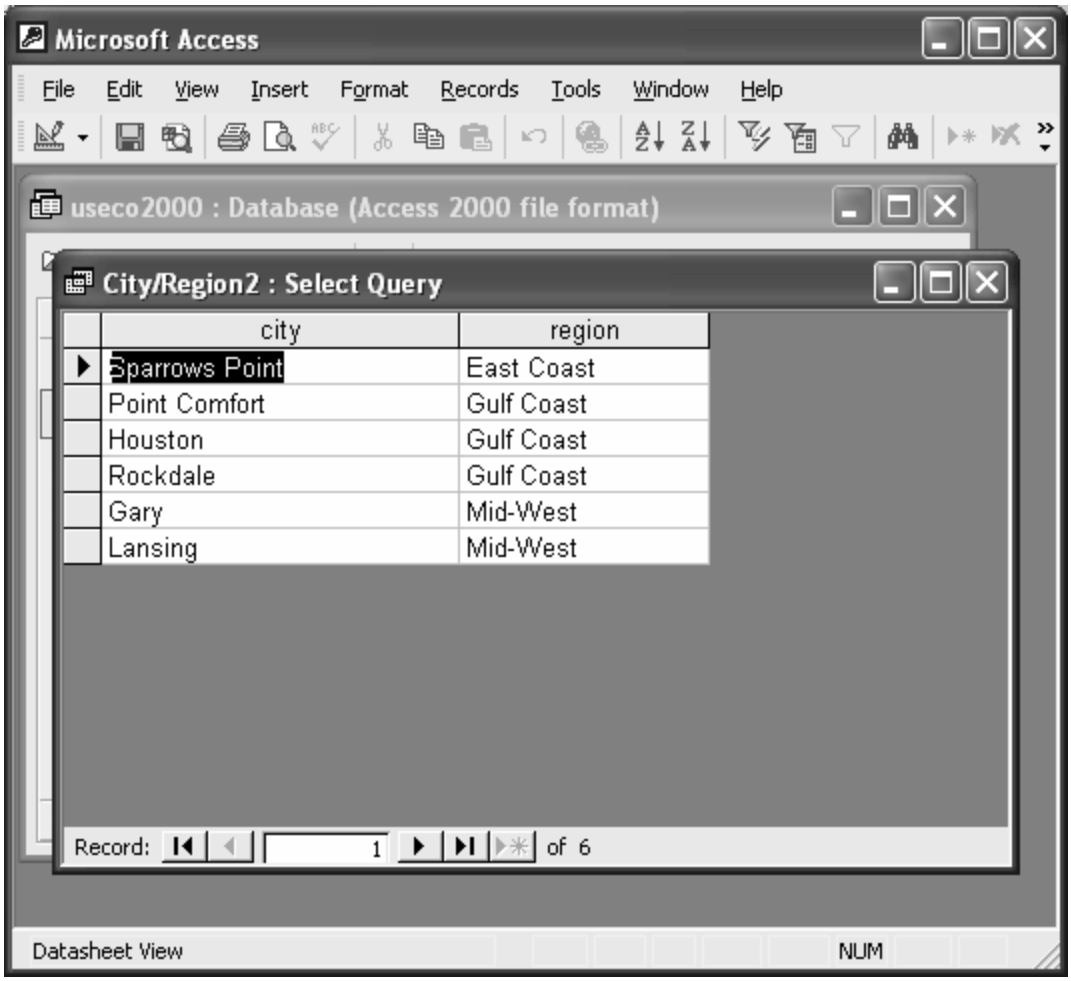


Figure 5.13 Results of the Query

This is the desired result - a table which shows the region in which each city is located.

As was mentioned above this seems like much too much work to find out that Houston is in the Gulf Coast region. However, once you have gained some facility with Access the point and click nature of the interface makes it an efficient way to develop queries. Moreover when you have a large database with many relationships and much data the power of the methods becomes apparent.

4. Examples

There are a number of examples of the use of this database given in Kendrick (1982b). Here it will suffice to describe a couple of them.

The first example is from the field on energy economics. Imagine that as the hurricane season approached emergency management officials want to know the amount of refining capacity on the Atlantic and Gulf of Mexico coasts. This roughly corresponds to the East-Coast and Gulf-Coast regions in the database so we clearly need to begin from the relationship

State location (state, region)

and from there we need to work backward with the

City location (city, state)

relationship and to the

Plant location (plant, city)

relationship. Then we need to cap this off by using the

Capacity (productive unit, plant, year, capacity level)

relationship.

If we do a join from plant to plant in the last two of these relationships and from city to city just above and then from state to state in the previous pair we will obtain a new relationship which we call

Regional capacity (productive unit, region, year, capacity level)

that will contain the information shown in Table 5.7 below.

		Region		
Productive Unit	East Coast	Gulf Coast	Mid-West	Units
Blast Furnace	2.0		2.5	mty
Steel Shop	2.35		2.8	mty
Rolling Mill	1.9		2.4	mty
Alumina Plant		0.8		mty
Aluminum Plant		1.1		mty
Primary Still		0.2		mbd
Catalytic Cracker		0.23		mbd
Auto Stamping			0.6	muy
Auto Assembly Line			0.6	muy

Table 5.7 Regional Capacity Relationship

Then from this table we can quickly see that that the Primary Still capacity (which is the best indicator of the refinery capacity) is 0.2 million barrels a day in the Gulf Coast region.

You can look at the implementation of this query in the database by going to the Query tab and then selecting the "prod unit/region/capacity" query. In particular, it is useful to single click this query and then click on the "Design" button so that you can see how the four relationships are used in the query and how they are linked by plant, city and state to create the desired relationship.

A second example comes from politics. From time to time Presidential politics in the U.S. are affected by difficulties in a particular industry. One example is the pressure that the U.S. auto industry has felt from imports at some times. Suppose that in a campaign a presidential candidate asks for a list of Democratic governors whose states have more than 10,000 people employed in the automobile and steel industries.

Clearly for this query we need to work back from

Party affiliation (governor, party)

to

State governors (governor, state)

to

City location (city, state)

to

Plant location (plant, city)

Then we need to make use of the

Plant employees (plant, union, number of employees)

relationship while also making use of the

Industry composition (plant, industry)

relationship.

If we do all the required joins properly we should obtain a relationship which we call

Employees by governor and party (industry, governor, party,
number of employees)

From this table we could then assembly the data required to answer the query.

These two examples provide an indication of how a relational database of the economy might be used to provide quick answers to a wide variety of questions. In most cases the answers to the questions could be provided in tables. In other cases the results of the queries would be time series or cross sections of data which would then subjected to further econometric analysis.

5. Experiments

A beginning experiment might be to implement the last of the two examples listed above; however, it is probably wise to begin with something simpler like the location of cities by region as described above but doing this without following the steps in the book.

Once you gain some confidence with doing simple joins you are encouraged to develop you own queries. If the existing relationships in the database are not sufficient, then you may want to add some additional ones in order to be able to answer richer and more interesting queries.

Finally, you might want to develop your own database with data from financial markets, labor relations or environmental economics as suits your interest. If you have had a summer job or an internship in a business or governmental agency you have likely made use of some databases and might want to try your hand at developing a similar database in Access or some other relational database software.

6. Further Reading

The classic book on relational database systems is Date (1977). To learn more about Access 2000 see Andersen (1999).

Appendix 5A

An Example U.S. Economy Database

This appendix contains the relationships from the example U.S. economy database from Kendrick (1982b).

		Process		
Commodity	Pig iron production	Steel production pig iron intensive	Steel production scrap intensive	Rolling flat steel products
Iron ore	-1.6			
Pig iron	1.0	-0.9	-0.7	
Scrap iron		-0.2	-0.4	
Liquid steel		1.0	1.0	-1.2
Scrap				0.2
Flat steel				1.0

	Alumina production	Aluminum production	Primary distillation	Catalytic cracking
Bauxite	-1.4			
Alumina	1.0	-1.2		
Aluminum		1.0		
Crude oil			-1.0	
Distillate			0.2	-1.0
Gasoline			0.3	0.6
Jet fuel			0.1	0.2

	Auto body stamping	Auto assembly
Flat steel	-1.2	
Aluminum	-0.2	
Auto bodies	1.0	-1.0
Automobiles		1.0

Table 5A.1 Input-Output

In the Input-Output relationship negative values indicate inputs and positive values outputs. Thus in the pig iron production process 1.6 tons of iron ore are used to produce 1.0 tons of pig iron. Then in the next column 0.9 tons of that pig iron is used along with 0.2 tons of scrap to produce a ton of liquid steel. The activity analysis vectors here follow in the tradition of Tjalling Koopmans (1951). Also for an introduction to use of activity analysis in economics see Kendrick (1996).

Thus a process is akin to a cook's recipe in that it provides a list of ingredients and how much is required of each as well as an indication of the final product or products. However unlike the usual recipe for a cake, a process may have a single input and multiple outputs, viz. the process above for primary distillation in an oil refinery where crude oil input is transformed into distillate, gasoline and jet fuel.

		Process		
Commodity	Pig iron production	Steel production pig iron intensive	Steel production scrap intensive	Rolling flat steel products
Pig iron - mty	86.8			
Liquid steel - mty		55.5	53.0	
Scrap - mty				18.0
Flat steel - mty				90.0

	Alumina production	Aluminum production	Primary distillation	Catalytic cracking
Alumina - mty	20.0			
Aluminum - mty		16.0		
Distillate - tby			1.46	
Gasoline - tby			2.19	2.43
Jet fuel - tby			0.73	0.73

	Auto body stamping	Auto assembly
Auto bodies - muy	9.5	
Automobiles - muy		9.35

Table 5A.2 Production

Note: mty = million tons per year
tby = trillion barrels per year
muy = million units per year

		Process		
Productive unit	Pig iron production	Steel production pig iron intensive	Steel production scrap intensive	Rolling flat steel products
Blast furnace	1			
Steel shop		1	1	
Rolling mill				1

	Alumina production	Aluminum production	Primary distillation	Catalytic cracking
Alumina plant	1			
Aluminum plant		1		
Primary still			1	
Catalytic cracker				1

	Auto body stamping	Auto assembly
Auto stamping plant	1	
Auto assembly plant		1

Table 5A.3 Capacity Use

The Capacity Use relationship simply tells the productive unit in which each process runs. Notice that substitute processes like the two for steel production both use the same productive unit, namely the steel shop.

plant	industry
Sparrows Point	steel
Rockdale	aluminum
ARCO-Houston	oil
Point Comfort	aluminum
Inland-Gary	steel
Lansing	automobile

Table 5A.4 Industry Composition

industry	sector
steel	primary metal
aluminum	primary metal
oil	petroleum and coal
automobile	transportation equipment

Table 5A.5 Sector Composition

productive unit	plant	capacity level	units
blast furnace	Sparrows Point	2.0	mty
blast furnace	Inland-Gary	2.5	mty
steel shop	Sparrows Point	2.35	mty
steel shop	Inland-Gary	2.8	mty
rolling mill	Sparrows Point	1.9	mty
rolling mill	Inland-Gary	2.4	mty
alumina plant	Point Comfort	0.8	mty
aluminum plant	Point Comfort	0.6	mty
aluminum plant	Rockdale	0.5	mty
primary still	ARCO-Houston	0.2	mbd
catalytic cracker	ARCO-Houston	0.23	mbd
auto stamping plant	Lansing	0.6	muy
auto assembly line	Lansing	0.6	muy

Table 5A.6 Capacity 1980

productive unit	plant	increment to capacity	units
alumina plant	Point Comfort	0.5	mty
aluminum plant	Point Comfort	0.4	mty
auto assembly line	Lansing	0.0	muy
auto stamping plant	Lansing	0.0	muy
blast furnace	Sparrows Point	0.5	mty
blast furnace	Inland-Gary	0.0	mty
catalytic cracker	ARCO-Houston	0.12	mbd
primary still	ARCO-Houston	0.1	mbd
rolling mill	Sparrows Point	0.4	mty
steel shop	Sparrows Point	0.5	mty
aluminum plant	Rockdale	0.0	mty
steel shop	Inland-Gary	0.0	mty
rolling mill	Inland-Gary	0.0	mty

Table 5A.7 Increment to Capacity 1981
Note: mty = million tons per year
mbd = million barrels per day
muy = million units per year

The table above is really about investment. However, it differs from the usual notion of investment which is a certain number of dollars spent on a new plant or pieces equipment. Rather the investment above is defined as an increment to capacity and is measured in units of the principal input or output of the productive unit. Thus the blast furnace capacity at Sparrows Point is increased by 0.5 million tons per year (an output) and the primary still at ARCO-Houston is increased by 0.1 million barrels per day (an input).

plant	corporation
Sparrows Point	United State Steel
Rockdale	ALCOA
ARCO-Houston	Atlantic Richfield Co.
Point Comfort	ALCOA
Inland-Gary	Inland Steel
Lansing	General Motors

Table 5A.8 Ownership

			Union			
Plant	OCAW	UAW	USA	IBEW	IBT	IAM
Sparrows Point			1.2		0.3	0.05
Rockdale			0.5	0.05		
ARCO-Houston	0.4				0.01	
Point Comfort			0.7		0.2	
Inland-Gary			0.4			
Lansing		1.2				

Table 5A.9 Plant Employees (in thousands of employees)

OCAW	Oil, Chemical and Atomic Workers
UAW	United Auto Workers
USA	United Steel Workers of America
IBEW	International Brotherhood of Electrical Workers
IBT	International Brotherhood of Teamsters
IAM	International Association of Machinists

plant	city
Sparrows Point	Sparrows Point
Rockdale	Rockdale
ARCO-Houston	Houston
Point Comfort	Point Comfort
Inland-Gary	Gary
Lansing	Lansing

Table 5A.10 Plant Location

city	state
Sparrows Point	Maryland
Rockdale	Texas
Houston	Texas
Point Comfort	Texas
Gary	Indiana
Lansing	Michigan

Table 5A.11 City Location

state	region
Maryland	East Coast
Texas	Gulf Coast
Indiana	Mid-West
Michigan	Mid-West

Table 5A.12 State Location

state	governor
Maryland	Harry Hughes
Texas	William P. Clements, Jr.
Indiana	Otis R. Bowen
Michigan	William G. Milliken

Table 5A.13 State Governors

governor	party
Harry Hughes	Democrat
William P. Clements, Jr.	Republican
Otis R. Bowen	Republican
William G. Milliken	Republican

Table 5A.14 Party Affiliation

Which relationships share a common domain?

Plant Location and City Location	are linked by	*city*
City Location and State Location and State Governor	are linked by	*state*
State Governor and Party Affiliation	are linked by	*governor*
Ownership and Plant	are linked by	*corporation*
Industry Composition and Sector Composition	are linked by	*industry*
Input-Output and Production	are linked by	*process*
	and	*commodity*
Production and Capacity Use	are linked by	*process*
Capacity Use and Capacity	are linked by	*productive unit*
Increment to Capacity and Capacity	are linked by	*productive unit*
	and	*plant*
Capacity and Plant Employees	are linked by	*plant*
Increment to Capacity and Plant Employees	are linked by	*plant*
Industry Composition and Plant Employees	are linked by	*plant*

Chapter 6

Thrift in GAMS

with

Genevieve Solomon

Many students face a tough financial problem – their expenses exceed their income. Thus they must work to supplement their income and/or borrow money from student loan funds. This familiar student situation provides a good setting to learn about dynamic personal financial planning models.

Some students have financial assets such as stocks and bonds; however many more students have few assets and substantial liabilities in the form of credit card debt and student loans. Thus we provide in this chapter a model in which a student can hold assets in either low interest bonds or higher interest stocks. Also, the student can hold some assets in a checking account while paying living expenses out of that account and depositing earnings from part time work into the account. If living expenses exceed earnings then the student must either draw down stock and bond assets or else borrow from a student loan fund at a low interest rate or from a credit card firm at a much higher interest rate.

We begin the chapter with a simple version of the model with only bonds, checking accounts and student loans and then advance to a more complex model later in the chapter.

1. The Mathematics of the Thrift Model

Consider a student who has a checking account as well as some money saved in government bonds. The dynamic equation for the bonds held by the student can be written as

(1) $Sb_{t+1} = (1+rb)Sb_t - Xbc_t + Xcb_t$

where

Sb = stock of bonds
rb = rate of interest on bonds
Xbc = transfer from bonds to checking account
Xcb = transfer from checking account to bonds

Thus the stock of bonds next period is equal to the stock of bonds this period multiplied by one plus the interest rate on the bonds minus bonds that are cashed in (Xbc) plus new bonds that are purchased (Xcb).

As is shown below, the proceeds from the sales of bonds (Xbc) are deposited in the student's checking account. Thus the equation for his or her checking account can be written as

(2) $Sc_{t+1} = (1+rc)Sc_t + Xbc_t - Xcb_t$

where

Sc = stock of funds in the checking account
rc = rate of interest on funds in the checking account

Likewise additional bonds can be purchased by withdrawing money from the checking account, Xcb.

The bond and checking accounts are both asset accounts for the student. Also, one can create a liability account in the form of a student loan equation that is the amount the students owes to the bank. This equation is

(3) $Ssl_{t+1} = (1+rsl)Ssl_t - Xcsl_t + Xslc_t$

where

Ssl = stock of student loans
rsl = rate of interest on student loans
$Xcsl$ = transfer from checking account to student loans
$Xslc$ = transfer from student loans to checking account

In this equation $Xcsl$ is the amount that the student withdraws from his or her checking account to repay student loans and $Xslc$ is the amount of money borrowed from the student's loan account to deposit in the student's checking account. Given these additional flows to and from the student's checking account we need to modify the checking account state equation from Eq. (2) above by including two more terms so that it becomes

$$Sc_{t+1} = (1+rc)Sc_t + Xbc_t - Xcb_t - Xcsl_t + Xslc_t \tag{4}$$

Also the student has a part-time job and deposits these wages, Wa, into the checking account and pays his or her living expenses, Le, from the account so we need to include two more terms in the equations so that Eq. (4) becomes

$$Sc_{t+1} = (1+rc)Sc_t + Xbc_t - Xcb_t - Xcsl_t + Xslc_t + Wa_t - Le_t \tag{5}$$

The model then consists of the bond equation (1), the checking account equation (5) and the student loan equation (3) and can be written in matrix form as

$$\begin{bmatrix} Sb \\ Sc \\ Ssl \end{bmatrix}_{t+1} = \begin{bmatrix} 1+rb & 0 & 0 \\ 0 & 1+rc & 0 \\ 0 & 0 & 1+rsl \end{bmatrix} \begin{bmatrix} Sb \\ Sc \\ Ssl \end{bmatrix}_t + \begin{bmatrix} -1 & 1 & 0 & 0 \\ 1 & -1 & -1 & 1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} Xbc \\ Xcb \\ Xcsl \\ Xslc \end{bmatrix}_t + \begin{bmatrix} 0 & 0 \\ 1 & -1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} Wa \\ Le \end{bmatrix}_t \tag{6}$$

or in vector difference equation form as

$$x_{t+1} = Ax_t + Bu_t + Cz_t \tag{7}$$

where the state vectors x_t, the control vector u_t and the exogenous vector z_t are defined as

$$x_t = \begin{bmatrix} Sb \\ Sc \\ Ssl \end{bmatrix}_t \qquad u_t = \begin{bmatrix} Xbc \\ Xcb \\ Xcsl \\ Xslc \end{bmatrix}_t \qquad z_t = \begin{bmatrix} Wa \\ Le \end{bmatrix}_t \tag{8}$$

and the matrices A, B and C are

$$(9) \qquad A=\begin{bmatrix} 1+rb & 0 & 0 \\ 0 & 1+rc & 0 \\ 0 & 0 & 1+rsl \end{bmatrix} \quad B=\begin{bmatrix} -1 & 1 & 0 & 0 \\ 1 & -1 & -1 & 1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \qquad C=\begin{bmatrix} 0 & 0 \\ 1 & -1 \\ 0 & 0 \end{bmatrix}$$

Difference equations models like Eq. (7) are frequently called "system" equations and are widely used in engineering and in economics to represent dynamic systems. Also, such models often have a criterion function that is optimized subject to the system equations.

A common form of the criterion function is the quadratic tracking function. This kind of criterion function is different than the usual utility maximization criterion used in consumer theory, the cost minimization criterion sometimes used in production theory or the terminal wealth maximization sometimes used in portfolio models. The quadratic tracking criterion function includes desired paths for the state variables and for the control variables and seeks to minimize the weighted squared separation between the desired paths and the optimal paths. For example, an individual may wish to save over the course of a lifetime for multiple purposes such as purchasing a house or car, paying for college educations for children and providing retirement income. Also, the individual may want to be sure to keep a target amount in a checking account. So the desired time path for savings accounts, stock and bond holdings and for checking account balances may be time-varying and have a complicated shape. Also, some goals may be more important than others and thus have higher weights attached to them.

A static version of the quadratic tracking criterion function with only a single state variable x and a single control variable u can be written

$$(10) \qquad J = w\left(x-\tilde{x}\right)^2 + \lambda\left(u-\tilde{u}\right)^2$$

where

J = criterion value
$\tilde{x}$ = desired value of the state variable
$\tilde{u}$ = desired value of the control variable
w = priority on the state variable
λ = priority on the control variable

Since w and λ are positive and the goal is to minimize J, one wants to have the state variable x be as close as possible to its desired value $\tilde{x}$ and the control variable u be as close as possible to its desired value $\tilde{u}$. Thus it is obvious that the criterion function in this case can be minimized by setting

(11) $$x = \tilde{x} \qquad u = \tilde{u}$$

However, this is usually not possible because the state and control variables are related to one another through the system equation. Thus, there is usually a trade-off between having x as close as possible to $\tilde{x}$ and u as close as possible to $\tilde{u}$. Furthermore this tradeoff is affected by the priority parameters w and λ. Thus if w is large and λ is small the optimal solution will be to set x close to $\tilde{x}$ and u not so close to $\tilde{u}$.

The priority parameters w and λ are also sometimes called penalty weights depending on whether one is thinking of them positively as priorities or negatively as penalties in a criterion function that is to be minimized. Both terms are used in this book and elsewhere in the literature.

When the state variable and the control variable are not scalars but rather vectors Eq. (10) can be written in vector-matrix form as

(12) $$J = \left(x - \tilde{x}\right)' W \left(x - \tilde{x}\right) + \left(u - \tilde{u}\right)' \Lambda \left(u - \tilde{u}\right)$$

where

x = state vector
u = control vector
$\tilde{x}$ = desired value of the state vector
$\tilde{u}$ = desired value of the control vector
W = diagonal priority matrix for the state vector
Λ = diagonal priority matrix for the control vector

Consider only the first term on the right hand side of Eq. (12). For the case at hand it can be written as

(13) $$\left(x - \tilde{x}\right)' W \left(x - \tilde{x}\right) = \begin{pmatrix} Sb - S\tilde{b} \\ Sc - S\tilde{c} \\ Ssl - Ss\tilde{l} \end{pmatrix}' \begin{bmatrix} wb & 0 & 0 \\ 0 & wc & 0 \\ 0 & 0 & wsl \end{bmatrix} \begin{pmatrix} Sb - S\tilde{b} \\ Sc - S\tilde{c} \\ Ssl - Ss\tilde{l} \end{pmatrix}$$

where

wb = priority for bonds
wc = priority for checking account
wsl = priority for student loan account

Taking the transpose of the first vector on the right hand side of Eq. (13) and doing the matrix vector multiplication of the remaining matrix and vector in that equation yields

$$(14) \qquad \left(x-\tilde{x}\right)' W \left(x-\tilde{x}\right) = \begin{pmatrix} Sb-S\tilde{b} & Sc-S\tilde{c} & Ssl-Ss\tilde{l} \end{pmatrix} \begin{bmatrix} wb\left(Sb-S\tilde{b}\right) \\ wc\left(Sc-S\tilde{c}\right) \\ wsl\left(Ssl-Ss\tilde{l}\right) \end{bmatrix}$$

or

$$(15) \qquad \left(x-\tilde{x}\right)' W \left(x-\tilde{x}\right) = wb\left(Sb-S\tilde{b}\right)^2 + wc\left(Sc-S\tilde{c}\right)^2 + wsl\left(Ssl-Ss\tilde{l}\right)^2$$

Since the W matrix is diagonal the quadratic form on the left hand side of Eq. (15) is equal to a weighted sum of squares of the differences between each state variable and its desired value with the weights being the respective priorities.

From a similar set of mathematical statements it could be shown that the quadratic form in the control variables in Eq. (12) is

$$(16) \qquad \begin{aligned} \left(u-\tilde{u}\right)' \Lambda \left(u-\tilde{u}\right) &= \lambda bc\left(Xbc-Xb\tilde{c}\right)^2 + \lambda cb\left(Xcb-Xc\tilde{b}\right)^2 \\ &\quad + \lambda csl\left(Xcsl-Xcs\tilde{l}\right)^2 + \lambda slc\left(Xslc-Xsl\tilde{c}\right)^2 \end{aligned}$$

where

λbc = priority on transfers from bonds to checking
λcb = priority on transfers from checking to bonds
λcsl = priority on transfers from checking to student loan
λslc = priority on transfers from student loan to checking

The priorities on the control variables in the λ parameters work analogously to those on the state variables, i.e. a large value for the priority indicates that the students wants to hold that control variable close to its desired values. Of course, what really matters is not the absolute values of the w and λ priorities but their values *relative* to one another. So the student who wants to assure that the state variables reach their desired values will

assign relatively high priorities to the state variables with the w parameters and relatively low priorities to the control variables with the λ parameters.

In summary, we can write the quadratic tracking criterion function for a single period as

$$J = (x - \tilde{x})' W (x - \tilde{x}) + (u - \tilde{u})' \Lambda (u - \tilde{u}) \tag{17}$$

However, we want to use this criterion function in a multiperiod model, therefore we need a dynamic version of Eq. (17) that can be written

$$J = \frac{1}{2}(x_N - \tilde{x}_N)' W_N (x_N - \tilde{x}_N) + \frac{1}{2}\sum_{t=0}^{N-1}\left[(x_t - \tilde{x}_t)' W (x_t - \tilde{x}_t) + (u_t - \tilde{u}_t)' \Lambda (u_t - \tilde{u}_t)\right] \tag{18}$$

It is customary to include the $\frac{1}{2}$ fractions in the criterion function so that when the derivatives of this quadratic function are taken the first order conditions will not include a two. Also, a distinction is made between the priorities on the state variables in the terminal time period N, i.e. W_N, and those in all other time periods, W. This permits different priorities to be attached to the state variables in the terminal period than in other periods. Also the control vector for the terminal period u_N does not appear in the criterion since it does not affect the state until period $N+1$ and that period is not included in the model.

In summary the dynamic control theory model seeks to find the control variables

$$(u_0, u_1, \cdots, u_{N-1})$$

that will minimize the criterion function

$$J = \frac{1}{2}(x_N - \tilde{x}_N)' W_N (x_N - \tilde{x}_N) + \frac{1}{2}\sum_{t=0}^{N-1}\left[(x_t - \tilde{x}_t)' W (x_t - \tilde{x}_t) + (u_t - \tilde{u}_t)' \Lambda (u_t - \tilde{u}_t)\right] \tag{19}$$

subject to the systems equations (from Eq. (7))

$$x_{t+1} = Ax_t + Bu_t + Cz_t \tag{20}$$

with the initial conditions

$$x_0 \text{ given} \tag{21}$$

The student financial model described above can be specified in this form using the state, control and exogenous variable vectors.

$$(22) \qquad x_t = \begin{bmatrix} Sb \\ Sc \\ Ssl \end{bmatrix}_t \qquad u_t = \begin{bmatrix} Xbc \\ Xcb \\ Xcsl \\ Xslc \end{bmatrix}_t \qquad z_t = \begin{bmatrix} Wa \\ Le \end{bmatrix}_t$$

and the matrices A, B and C

$$(23) \qquad A = \begin{bmatrix} 1+rb & 0 & 0 \\ 0 & 1+rc & 0 \\ 0 & 0 & 1+rsl \end{bmatrix} \quad B = \begin{bmatrix} -1 & 1 & 0 & 0 \\ 1 & -1 & -1 & 1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \qquad C = \begin{bmatrix} 0 & 0 \\ 1 & -1 \\ 0 & 0 \end{bmatrix}$$

with the exogenous variable $(z_0, z_1, \cdots, z_{N-1})$ and initial state x_0 given.

2. The Evanchik Model

A model of this form was developed by Michael Evanchik (1998) when he was an undergraduate in the Computational Economics class at the University of Texas. Evanchik's model is slightly more complicated than the model described above in that it includes three types of assets rather than two by adding equities to the bond and checking accounts used above. Also, the model includes two liabilities rather than one since the student can borrow not only from a student loan account but also from that most popular source of student support - a credit card account. Thus the state vector for this model has five variables, i.e.

$$(24) \qquad x_t = \begin{bmatrix} Sb \\ Se \\ Sc \\ Scc \\ Ssl \end{bmatrix}_t$$

where

Sb = stock of bonds
Se = stock of equities
Sc = stock of funds in the checking account
Scc = stock of credit card loans
Ssl = stock of student loans

Correspondingly, the control vector includes more elements to permit a variety of transfers among these accounts. The control vector is

$$u_t = \begin{bmatrix} Xbe \\ Xbc \\ Xbcc \\ Xbsl \\ Xec \\ Xecc \\ Xesl \\ Xcacc \\ Xcsl \\ Xccsl \end{bmatrix} \tag{25}$$

where

Xbe = transfer from bonds to equities
Xbc = transfer from bonds to checking account
Xbcc = transfer from bonds to credit card account
Xbsl = transfer from bonds to student loans
Xec = transfer from equities to checking account
Xecc = transfer from equities to credit card account
Xesl = transfer from equities to student loans
Xcacc = transfer from checking account to credit card account
Xcsl = transfer from checking account to student loan account
Xccsl = transfer from credit card account to student loan account

There is one anomaly in the variable naming scheme above which has been introduced to eliminate a source of confusion. The transfer which would have been labeled $Xccc$ to be consistent with all of the rest of the variable names has instead been labeled $Xcacc$ to make it clear that the transfer is from the checking account to the credit card account rather than vice versa.

A GAMS version of the model was created by one of the authors of this chapter, Genevieve Solomon, while she was a student in the same class a couple of years after Michael Evanchik. She added a third exogenous variable to the two already in the Evanchik model so that the exogenous variable vector for the model in this chapter is

$$z_t = \begin{bmatrix} Wa \\ Le \\ Sh \end{bmatrix}_t \tag{26}$$

where

Wa = wages
Le = living expenses
Sh = scholarship

The GAMS version of the thrift model has the useful property that it is possible to put explicit upper bounds on the state and control variables. For example there are frequently upper bounds on how much money a student can borrow per semester from the student loan organization. Also, credit cards frequently have upper bounds on the amount that a student can borrow.

3. The Model in GAMS

The GAMS program corresponding to the thrift problem is available at the book web site. The first step for the GAMS version of the model is to define the sets. There are four sets: state variables, control variables, exogenous variables and the time horizon. These sets are declared and defined in GAMS as follows:

```
Sets n  states     / Sb, Se, Sc, Scc, Ssl /
     m  controls   /Xbe, Xbc, Xbcc, Xbsl, Xec,
                    Xecc, Xesl, Xcacc, Xcsl, Xccsl/
     k  exogenous  / Wa, Le, Sh /
     t  horizon    / 2000, 2001, 2002, 2003, 2004 /
```

Notice here how sets are specified in the GAMS language. In this model there are five state variables. The set of states could be specified in mathematics as

$$N = \{Sb,\ Se,\ Sc,\ Scc,\ Ssl\}$$

The equivalent GAMS statement would be

```
n = / Sb, Se, Sc, Scc, Ssl /
```

GAMS has forward slashes as set delimiters while mathematics has braces. This means that you should be very careful not to use forward slashes in a GAMS model in text statements like "`dollars/ton`" since the slash will confuse the GAMS compiler and may result in an error. Also, we include in the statement for the set of state variables the word "state" which is the text that is associated with the set `n`. Thus the complete statement in GAMS for the set `n` is

```
n  states     / Sb, Se, Sc, Scc, Ssl /
```

Also, similar statements are used to declare and define the sets of controls variables `m`, exogenous variables `k` and the set of time periods `t`.

Next three subsets of the time set are declared. In many computer languages a distinction is made between "declaring" an element and "defining" it. That distinction is also used here since the statements below are used to declare three sets that will be defined later when the elements in the set will be determined.

```
tu(t) control horizon
ti(t) initial period
tz(t) terminal period ;
```

The control horizon, initial period and the terminal period sets will be important later for the equations. Also, in these three statements the `(t)` is used to indicate that the preceding set, viz. `tu`, is a subset of the set `t`.

Here in the body of this chapter we will introduce the parts of the GAMS statement of the model one section at a time. Later you may want to look at the entire GAMS statement of the model that is in Appendix 6A at the end of the chapter.

Next tables are created to represent the matrices in the systems equations, Eq. (20)

A (one plus interest rates) (5x5)
B (direction of transfers) (5x10)
C (exogenous variable signs) (5x3)
w (state variable penalty matrix) (5x5)
wn (state variable penalty matrix for the terminal period) (5x5)
lambda (control variable penalty matrix) (10x10)

In doing this we also need an "alias" statement as follows:

```
Alias (n,np), (m,mp) ;
```

This alias statement simply makes a copy of the set n and calls it np (n prime) and of the set m and calls it mp (m prime). This alias is necessary for setting up summations for matrix operations in GAMS. Also the aliases will be needed when matrices are transposed in later equations.

The next part of the input defines the time subsets, tu, ti and tz of the full time set t. The first of these, tu, is the set of all time periods other than the terminal period and is defined with the GAMS statements

```
tu(t) = yes$(ord(t) lt card(t));
```

This statement makes use of two GAMS keywords ord and card that are operators defined on sets. "card" is an abbreviation for cardinal, which is the number of elements in the set. Consider a set S in mathematics

$$S = \{a, b, c, d\}$$

The cardinality of this set is four since it has four elements. In contrast "ord" is an abbreviation for ordinal, which represents the ordinal position of each element in the set. Thus the element c is in the third ordinal position in the set above.

So the GAMS statement defining the set tu can be read, "tu is the set of elements whose ordinal position in the set t are strictly less than the cardinality of the set". Thus, recalling that the set t is

```
tu =    / 2000, 2001, 2002, 2003, 2004 /
```

we see that the set `tu` is

```
tu =    / 2000, 2001, 2002, 2003 /
```

i.e. it is all the elements in the set `t` except the last element.

The second of the subsets of `t` is the set `ti`, which is the initial time period only. It is defined with the GAMS statement

```
ti(t) = yes$(ord(t) eq 1);
```

Thus the set `ti` contains the element which is in the first position in the set `t`, namely "`2000`".

The third subset of t is defined with the GAMS statement

```
tz(t) = not tu(t);
```

Thus `tz` is the set of all elements in the set `t` that are not in the set `tu` and that is only the last element, namely "`2004`".

Finally, just as a check, the elements of the full set and the three subsets are displayed in the output file with the statement

```
Display t, ti, tz, tu;
```

When doing set manipulations in GAMS it is useful to display the results as a check against errors.

Once the sets are specified, then the data can be input using the "table" and "parameter" keywords as shown below. Consider first the use of the table keyword to input the A matrix.

```
Table a(n,np)  state vector matrix
        Sb      Se      Sc       Scc      Ssl
Sb     1.05
Se              1.10
Sc                      1.01
Scc                              1.13
Ssl                                       1.03
```

Observe that the parameter "a" is followed by the sets over which it is defined, i.e. it is written as "a(n,np)". It is not necessary to include the sets here, however it is a useful precaution because when the sets are provided the GAMS complier can check to be sure that all the element names used in the input of the table do indeed belong to the appropriate sets for the rows and columns of the table. Thus if the user misspells an element in the table input GAMS will issue a warning.

Following the name of the table and its set is a line of text, i.e.

```
state vector matrix
```

The ability to use text like this phrase makes GAMS statements much easier to read and understand. The convention in GAMS is that the absence of an explicit data entry in a table results in that element of the matrix being set to zero. So all the elements in the A matrix others than those on the diagonal are set equal to zero.

Recall that the diagonal elements in the A matrix are one plus the appropriate interest rate. So the interest rates on bonds is 5 percent, on equities is 10 percent and on checking accounts is only 1 percent. (Of course bonds and equities have greater risk than checking accounts. Comparative risk is not addressed in this model but is included in the models on portfolio selection used later in this book.) One way to alter the model to better represent the financial condition of a given individual is to change the interest rates in this table to reflect the times and the person's own financial situation.

The input table for the B matrix in GAMS is

```
Table b(n,m)  control vector matrix
       Xbe   Xbc  Xbcc  Xbsl  Xec  Xecc  Xesl
Sb      -1    -1   -1    -1
Se       1                      -1    -1    -1
Sc              1                1
Scc                 -1                -1
Ssl                         -1                -1

+
       Xcacc Xcsl Xccsl
Sb
Se
Sc       -1   -1
Scc      -1          1
Ssl           -1    -1
```

This table is too wide to fit on a single page so the "+" symbol is used between the two parts of the table in GAMS to indicate that additional columns of the table are input in a second set of rows.

Consider first only the first four columns of the B matrix, which are all transfers out of the bond account. The first two columns (Xbe and Xbc) are transfers to other assets, i.e. bonds to equities and bonds to the checking account. Thus there is a minus one in the bonds row and a plus one in the equities row and the checking account row respectively. The next two columns ($Xbcc$ and $Xbsl$) are transfers from an asset account (bonds) to liability accounts (credit card and student loan respectively) so there is still a minus one in the bond row. However, there are also minus ones in the credit card and student loan rows since these transfers have the effect of decreasing the amount of credit card debt or of student loan debt through the action of selling bonds to payoff some of these loan amounts.

The input table for the C matrix in GAMS is

```
Table c(n,k)  exogenous vector matrix
        Wa      Le      Sh
Sb
Se
Sc      1       -1       1
Scc
Ssl
```

The only entries in this matrix are in the checking account row since wages and scholarships are deposited in this account and living expenses are withdrawn from it.

The criterion function priorities (penalty matrices) are input next. The matrix for the priorities for the state vectors for all periods other than the terminal period, W, is

```
Table w(n,np)  state vector matrix penalty matrix
        Sb     Se     Sc      Scc     Ssl
Sb     100
Se             100
Sc                    400
Scc                           200
Ssl                                    0
```

This is a diagonal $n \times n$ matrix; however the set `n` and its alias `np` are used. This is not essential here but it makes it easier to understand the notation that is used later in the specification of the criterion function in GAMS. Since the priority for the checking account is set high at `400` one would expect to observe in the solution that the checking account state variable Sc will track more closely to its desired value $S\tilde{c}$ than will other state variables to their respective desired values.

Next comes the input for the W_N matrix that is the state variable priority matrix for the terminal period N.

```
Table wn(n,np)  terminal state vector matrix penalty matrix
        Sb     Se     Sc      Scc     Ssl
Sb      200
Se             200
Sc                    800
Scc                           200
Ssl                                    1
```

These values are set twice as high as the priorities for the state variables in all other time periods.

This is followed by the input for the Λ matrix that is the control variable priority matrix for all time periods.

```
Table lambda(m,mp) lambda matrix
       Xbe  Xbc  Xbcc  Xbsl  Xec  Xecc Xesl
Xbe    20
Xbc          1
Xbcc               20
Xbsl                    20
Xec                          20
Xecc                               20
Xesl                                    20

+
         Xcacc     Xcsl     Xccsl
Xcacc      1
Xcsl                1
Xccsl                         20
```

All of these priorities are set to 20 except for those for transfers from the bond account to the checking account, from the checking account to the credit card account and from the checking account to the student loan account. Thus these three transfers are permitted to deviate more from than their desired paths than are the other transfers.

Since the desired path for the state vector $\tilde{x}_t$ (x_t with a tilde over it) is time varying it can be conveniently input with a `table` statement.

```
Table xtilde(n,t)  state vector desired paths

      2000        2001         2002       2003        2004
Sb
Se
Sc    1000        1000         1000        1000        1000
Scc   2000        2000         2000        2000        2000
Ssl
```

Recall the GAMS convention that a blank input in a table is treated as a zero. Therefore the desired path for bonds, equities and student loans are all set to zero. It is desired that the checking account hold steady at about $1,000 and the student's credit card debt also hold steady but at around $2,000.

Also the desired path for the control vector $\tilde{u}_t$ is time varying, so it can likewise be input with a `table` statement.

```
Table utilde(m,t)  control vector desired paths

          2000     2001     2002     2003
Xbe
Xbc
Xbcc
Xbsl
Xec
Xecc
Xesl
Xcacc
Xcsl
Xccsl
```

Since this table in entirely blank the desired values for all the transfers in all time periods are set to zero.

After the matrices in the systems equations and criterion function are input with `table` statements, the next step is to input the initial period values of the state vector. Since this is a vector it can be input with a `parameter` statement.

```
Parameter
        xinit(n)    initial value   /
                  Sb 4000
                  Se    0
                  Sc 1000
                  Scc   0
                  Ssl   0 /
```

As a first approximation, one can think of the `parameter` keyword in GAMS as the way to input a vector of data and the `table` keyword as the way to input a matrix. Thus the "`xinit(n)`" parameter was used above to input the vector that contains the initial values of the state variables.

So the student begins with $4,000 in bonds, no equities and $1,000 in his or her checking account. Also the student does not initially have any credit card debt or student loan debt. This vector is particularly useful in the experiments with this model since the most obvious thing to do to tailor the model to an individual's personal situation is to change the initial values for the state variables.

Next comes the input for the exogenous variables z_t that are time varying. Since this is a vector that changes over time it can be input with a `table` statement , i.e.

```
Table z(k,t)  exogenous variables

      2000        2001         2002        2003        2004
Wa   15000       15000        15000       15000       15000
Le   20000       20000        20000       20000       20000
Sh       0           0            0           0           0
```

The student has wages from his or her part time job of $15,000 a year and has living expenses of $20,000 a year and no scholarship help. Therefore the student must borrow approximately $5,000 a year or draw down his or her assets. Like the initial conditions this table is an obvious place for tailoring the model to an individual either by altering the wages, living expenses and scholarship aid over time or inputting a pattern more closely related to the individual own situation with respect to these exogenous variables.[10]

The variables are the next thing to be assigned in the GAMS program.

```
Variables
      u(m,t)    control variable
      j         criterion ;
Positive Variables
      x(n,t)   state variable ;
```

Aside from the criterion variable `j` the only two sets of variables in the model are the control variables `u` and the state variables `x`. The control variables can be either positive or negative. For example if the variable *Xbc* is positive it is a transfer from the bond account to the checking account and if it is negative it is a transfer from the checking account to the bond account. On the other hand, the state variables must be positive. For example `Scc` is a liability account and is the credit card debt of the student. If this amount were negative it would mean that the student was lending money to his or her credit card company. While some students might like to do that at 13 percent, it is unlikely that the credit card company would be willing to enter into such a deal. Therefore the restriction that the state variables must be positive is imposed in GAMS with the key words `Positive Variables`.

[10] Thanks to one or our students, Vivek Shah, for helping to develop the time-varying exogenous variable version of the thrift model.

Next the equations are declared in GAMS with the statements

```
Equations   criterion       criterion definition
            stateq(n,t)     state equation ;
```

So the only sets of equations in this model are the single equation for the criterion function and the $n \times t$ state equations. Since there are five state variables and five time periods then the model will have 25 state equations.

Next the equations are defined, beginning with the criterion function. Recall from Eq. (19) that this equation is in three parts, the state variables for the terminal period, the state variables for all other time periods and the control variables for all other time periods, i.e.

$$(19) \qquad J = \frac{1}{2}\left(x_N - \tilde{x}_N\right)' W_N \left(x_N - \tilde{x}_N\right) + \frac{1}{2}\sum_{t=0}^{N-1}\left[\left(x_t - \tilde{x}_t\right)' W \left(x_t - \tilde{x}_t\right) + \left(u_t - \tilde{u}_t\right)' \Lambda \left(u_t - \tilde{u}_t\right)\right]$$

Consider for the moment only the first part, i.e. the state variables in the terminal time period. This can be written with indices, rather than in vector-matrix form, as

$$(27) \qquad J = \frac{1}{2}\sum_{i \in I}\sum_{j \in J}\left(x_{iN} - \tilde{x}_{iN}\right) w_{iN} \left(x_{jN} - \tilde{x}_{jN}\right)$$

This, in turn, can be represented in GAMS with the statement

```
criterion..
j =e=
 .5*sum( (tz,n,np),
 (x(n,tz) - xtilde(n,tz))*wn(n,np)*(x(np,tz) - xtilde(np,tz)) )
```

This code begins with the name of the equation, `criterion`, and the two dots (`..`) following the name signal to the GAMS compiler that the name has been completed and the equation itself is to follow.

The sum in the mathematics in Eq. (27) is over the two sets I and J while the sum in GAMS is over three sets (`tz,n,np`). Since the set `tz` in GAMS has only a single element, namely the terminal period N in fact this sum in GAMS is really only over two sets. Recall that `n` is the set for the state variables, which are the stocks of bonds,

equities, checking account, credit card account and student loans. Also the set `np` in GAMS is the `alias` of the set `n`, i.e. it is a copy of the set.

The second part of the criterion function, namely the state variable for all periods other than the terminal period is written in mathematics with indices as

$$\frac{1}{2}\sum_{t\in Tu}\sum_{i\in I}\sum_{j\in J}\left(x_{it}-\tilde{x}_{it}\right)w_{it}\left(x_{jt}-\tilde{x}_{jt}\right) \tag{28}$$

where

Tu = set of all time periods except the terminal period

and this is written in GAMS as

```
+ .5*sum( (tu,n,np),
  (x(n,tu) - xtilde(n,tu))*w(n,np)*(x(np,tu) - xtilde(np,tu)) )
```

The sum here is indeed over the three sets, namely, `tu`, the set of all time periods other than the terminal period, and `n` and `np` the state variable set and its alias.

The final piece of the criterion function is written in mathematics with indices as

$$\frac{1}{2}\sum_{t\in Tu}\sum_{i\in I}\sum_{j\in J}\left(u_{it}-\tilde{u}_{it}\right)\lambda_{it}\left(u_{jt}-\tilde{u}_{jt}\right) \tag{29}$$

and in GAMS as

```
+ .5*sum( (tu,m,mp),
  (u(m,tu) - utilde(m,tu))*lambda(m,mp)*(u(mp,tu) - utilde(mp,tu)) ) ;
```

So this sum is over the time period set `tu` and also over the control variable set `m` and its alias `mp`.

In addition to the criterion function the only other equations in the model are those in the set of system equations. Recall that these equations are written mathematically as

$$x_{t+1} = Ax_t + Bu_t + Cz_t \tag{20}$$

In GAMS they are written

```
stateq(n,t+1)..
  x(n,t+1) =e=
        sum(np,(a(n,np)*x(np,t))) +
        sum(m, (b(n,m)*u(m,t))) +
        sum(k, (c(n,k)*z(k,t)));
```

The name of the equation in GAMS is `stateq` and it is defined for the sets `n` and `t+1`. Recall that the set t is

(30) $$T = \{2000, 2001, 2002, 2003, 2004\}$$

Then the set `t+1` in GAMS is defined as the set `t` less the first element in the set, namely

(31) $$\{2001, 2002, 2003, 2004\}$$

Thus `stateq` is defined over all time periods in the model except for the first time period.

Finally it is necessary to specify the initial conditions for the state variables of the model with the GAMS statement

```
x.fx(n,ti) = xinit(n);
```

The suffix `.fx` is used in GAMS as an abbreviation for "fixed" . In this statement then the state vector `x` is fixed in period `ti`, which is the initial period, to the values in the vector `xinit` which is the parameter vector that contains the initial conditions for the model.

Though it is not shown in the present version of the model, upper bounds on the credit card account and the student loan account can be included in GAMS statement. This is done at the end of the equations and before the `Solve` statement with upper bounds on variables. An example of this is shown below.

```
x.up('Scc',t)=5000;
x.up('Ssl',t)=7000;
```

The `x.up` means the upper bound for the variable `x`. So `x.up('Scc',t)` is an upper bound on the credit card and the `x.up('Ssl',t)` is the upper bound on the student loan. One can change these bounds to fit his or her own financial situation.

Next a name is assigned to the model while also indicating the equations that are included with the statement, in this case all of the equations

```
Model track  /all/ ;
```

This is followed by a statement directing that the model be solved with a nonlinear programming solver by minimizing j, the criterion function, i.e.

```
Solve track minimizing j using nlp ;
```

For an introduction to nonlinear optimization solvers see App. F and for a discussion of the stacking method in GAMS that is used for indexed model like this one see App. H.

Finally a table of the results is obtained with the use of the statement

```
Display x.l, u.l ;
```

The suffix "`.l`" on the variables `x` and `u` is not the number one but rather the letter `l` and is used to indicate the activity level of the variable.

Though it is not shown in the present model it is also possible to solve the model by maximizing terminal wealth.

4. Results

As was discussed above, Appendix A at the end of the book contains instructions on running GAMS. Recall from that discussion that examining the results from a GAMS run can seem complicated at first because the GAMS output files contain a substantial amount of information about the structure of the model and its solution. However, it is simple enough to jump around in the file to examine the key parts.

First locate the Solve Summary part of the output. To do this search in the editor for the string "`SOLVER STATUS`". When you do so you will see a section of the output that looks like

```
               S O L V E      S U M M A R Y

     MODEL   track               OBJECTIVE  j
     TYPE    NLP                 DIRECTION  MINIMIZE
     SOLVER  CONOPT              FROM LINE  166

**** SOLVER STATUS     1 NORMAL COMPLETION
**** MODEL STATUS      2 LOCALLY OPTIMAL
**** OBJECTIVE VALUE        1377722382.8446
```

As was discussed earlier, each time after you solve a GAMS model you should check this section of the output to be sure that the model was solved successfully. The words "NORMAL COMPLETION" here indicate that is the case. If the solution procedure was not successful you will find words like "INFEASIBLE" or "UNBOUNDED".

Next skip down the output across the sections labeled "---- EQU" until you get to the section labeled "---- VAR x state variable" which looks like

```
---- VAR x  state variable

            LOWER     LEVEL     UPPER    MARGINAL

Sb .2000  4000.000  4000.000  4000.000 4.2771E+5
Sb .2001       .      463.607     +INF       .
Sb .2002       .       17.262     +INF  -1.142E-9
Sb .2003       .        2.981     +INF  -3.220E-9
Sb .2004       .          .       +INF  19101.665
Se .2000       .          .         .   24584.643
Se .2001       .      405.341     +INF       EPS
Se .2002       .       39.804     +INF  -1.291E-9
Se .2003       .       12.292     +INF  -2.918E-9
Se .2004       .          .       +INF  18069.230
Sc .2000  1000.000  1000.000  1000.000 23323.625
Sc .2001       .     1107.581     +INF  -1.062E-9
Sc .2002       .     1001.230     +INF  -1.495E-9
Sc .2003       .      998.805     +INF  -3.380E-9
Sc .2004       .      975.797     +INF       EPS
Scc.2000       .          .         .   -4.280E+5
Scc.2001       .     1766.529     +INF  -9.31E-10
Scc.2002       .     1984.219     +INF       .
Scc.2003       .     1991.238     +INF       .
Scc.2004       .     2096.163     +INF       EPS
Ssl.2000       .          .         .   -2.421E+4
Ssl.2001       .          .       +INF  39980.703
Ssl.2002       .     4018.936     +INF  1.1896E-9
Ssl.2003       .     9371.548     +INF  3.2469E-9
Ssl.2004       .    14850.699     +INF       .
```

The interesting part here is the activity level of the shipment variables x in the column labeled "LEVEL". This shows, among other things, that there was 4000 in bonds in time period 2000 and 463 in bonds in time period 2001. This is the solution of the model that we were looking for. These same results are shown a little further down in the output in a section labeled "---- 169 VARIABLE x.L state variable" which is the result of the display statement in the GAMS input. That output is shown below.

```
----    169 VARIABLE  x.L  state variable

           2000        2001        2002        2003        2004

Sb     4000.000     463.607      17.262       2.981
Se                  405.341      39.804      12.292
Sc     1000.000    1107.581    1001.230     998.805     975.797
Scc                1766.529    1984.219    1991.238    2096.163
Ssl                            4018.936    9371.548   14850.699
```

This table is somewhat easier to read than the default output and thus you can see the reason that most GAMS input files end with a series of display statements. These tables are easily found since they are at the end of the long GAMS output so the user can quickly scroll to the bottom of the file and find the key results. However, they will be there only if you remember to add a display statement at the end of the GAMS input statement. So in summary, when looking at the GAMS output you should first check to be sure that the problem was solved satisfactorily. Then focus on the variables section.

Recall that the student starts with $4,000 in bonds and $1,000 in her checking account and that the student has $15,000 per year in wages and $20,000 in living expenses. Also, the desired path for the checking account is $1,000 and for the credit card account is $2,000.

As the table of state variable results over time above shows, the bond account is drawn down in the first two periods and the student also borrows roughly $2,000 on her credit card. Then in the third time period borrowing begins from the student loan account and reaches about $15,000 by the last period. So in order to finance the $5,000 shortfall each year over the four year period the student cashes in $4,000 in bonds, borrows $2,000 on her credit card and borrows about $15,000 from the student loan fund. Meanwhile the student continues to hold about $1,000 in her checking account in all time periods.

The transfers that are necessary to accomplish these results are shown in the control variable time paths below.

```
----    169 VARIABLE  u.L  control variable

             2000        2001        2002        2003

Xbe       201.984    -124.453     -55.584     -51.622
Xbc      3296.649     721.808     275.338     260.756
Xbcc       82.078      20.062      11.086       6.544
Xbsl      155.683    -147.892    -215.696    -212.548
Xec       -37.151     160.543      69.351      64.660
Xecc     -119.906     144.515      66.670      58.166
Xesl      -46.301     -23.439    -160.112    -160.927
Xcacc   -1655.095    -320.572     -53.610    -129.867
Xcsl     -182.988   -3679.651   -4589.264   -4511.721
Xccsl      73.605    -167.954    -226.783    -219.093
```

There is a transfer of $3,296 from the bond account to the checking account, `Xbc`, in the first time period followed by a transfer of $721 in the second period. Also there is a negative transfer of about -$1,600 from the checking account to the credit card account, `Xcacc`, in the first time period. So this is actually a transfer of about $1,600 from the credit card account to the student's checking account. This in turn is followed by a similar transfer of about $300 in the second time period.

Also, the borrowing from the student loan fund begins in the second time period when about $3,600 is transferred from the student loan account to the checking account via the variable `Xcsl`. This is followed by transfers of approximately $4,500 of a similar nature in the third and fourth time periods.

5. Experiments

The most useful experiment to do with this model is for the student to use it to take a rough look at his or her own finances during college and graduate school. The most important steps to accomplish this are to change the initial conditions in `xinit` and the wages, living expenses and scholarship aid in the exogenous variables, `z`. Also, the interest rates faced by the student are likely to be different than those used above and should be modified to be realistic.

Finally the student may have very different desired paths for the state variables. For example she may want to keep the bond account constant over the time horizon

covered by the model or she may want to limit credit borrowing to a smaller amount than was used in the model above.

There are other interesting experiments one can do simply by including bounds on the variables. For example, one can put a lower bound on the checking account. Some students have accounts where they are supposed to keep at least a minimum balance, viz $800. Thus one can place a lower bound of $800 on the checking account. In GAMS code the bound would look like the following:

```
x.lo('Sc',t)=800
```

More complicated experiments are to increase the time horizon covered by the model say to about ten periods and thus to cover not only years in college but the first years of employment when paying back student debt may become a priority. Another possibility is to solve the model by maximizing terminal wealth instead of minimizing the criterion function. Some of these last three experiments require changes in the specification of the model and are more difficult; however, they are a good way to learn more about GAMS and about financial planning.

Appendix 6A

The GAMS Statement of the Thrift Model

```
*Student Finance Model in GAMS
*By Genevieve Solomon
*This version also has some modifications by David Kendrick


Sets    n states  / Sb, Se, Sc, Scc, Ssl /
        m controls  /Xbe,Xbc,Xbcc,Xbsl,Xec,Xecc,Xesl,Xcacc,Xcsl,Xccsl /
        k exogenous  / Wa, Le, Sh /
        t horizon  / 2000, 2001, 2002, 2003, 2004 /

        tu(t) control horizon
        ti(t) initial period
        tz(t) terminal period ;

Alias (n,np), (m,mp) ;

tu(t) = yes$(ord(t) lt card(t));
ti(t) = yes$(ord(t) eq 1);
tz(t) = not tu(t);
Display t, ti, tz, tu;

Table a(n,np)  state vector matrix

        Sb     Se     Sc      Scc     Ssl
Sb      1.05
Se              1.10
Sc                     1.01
Scc                            1.13
Ssl                                    1.04

Table b(n,m)  control vector matrix

       Xbe      Xbc     Xbcc      Xbsl      Xec      Xecc      Xesl
Sb      -1       -1       -1        -1
Se       1                                  -1        -1        -1
Sc                1                          1
Scc                       -1                           -1
Ssl                                  -1                           -1

+
```

```
       Xcacc     Xcsl     Xccsl
Sb
Se
Sc      -1        -1
Scc     -1                 1
Ssl              -1       -1


Table c(n,k)  exogenous vector matrix

        Wa      Le      Sh
Sb
Se
Sc      1       -1       1
Scc
Ssl

Table w(n,np)  state vector matrix penalty matrix

        Sb     Se     Sc      Scc     Ssl
Sb     100
Se             100
Sc                    400
Scc                           200
Ssl                                   0


Table wn(n,np)  terminal state vector matrix penalty matrix

        Sb     Se     Sc      Scc     Ssl
Sb     200
Se             200
Sc                    800
Scc                           200
Ssl                                   1

Table lambda(m,mp) lambda matrix

       Xbe      Xbc     Xbcc      Xbsl      Xec      Xecc      Xesl
Xbe    20
Xbc             1
Xbcc                    20
Xbsl                              20
Xec                                       20
Xecc                                               20
Xesl                                                         20
```

```
+
         Xcacc     Xcsl     Xccsl
Xcacc      1
Xcsl                 1
Xccsl                        20


Table xtilde(n,t)  state vector desired paths

      2000        2001         2002        2003        2004
Sb
Se
Sc    1000        1000         1000        1000        1000
Scc   2000        2000         2000        2000        2000
Ssl


Table utilde(m,t)  control vector desired paths

          2000     2001     2002     2003
Xbe
Xbc
Xbcc
Xbsl
Xec
Xecc
Xesl
Xcacc
Xcsl
Xccsl

Parameter
        xinit(n)   initial value   /
             Sb  4000
             Se     0
             Sc  1000
             Scc    0
             Ssl    0 /

Table z(k,t)  exogenous variables

      2000        2001         2002        2003        2004
Wa   15000       15000        15000       15000       15000
Le   20000       20000        20000       20000       20000
Sh       0           0            0           0           0
```

```
Variables               u(m,t)    control variable
                        j         criterion ;

Positive Variables      x(n,t)   state variable ;


Equations       criterion       criterion definition
                stateq(n,t)     state equation ;

criterion..

j =e=
      .5*sum( (tz,n,np),
      (x(n,tz) - xtilde(n,tz))*wn(n,np)*(x(np,tz) - xtilde(np,tz)) ) +

      .5*sum( (tu,n,np),
      (x(n,tu) - xtilde(n,tu))*w(n,np)*(x(np,tu) - xtilde(np,tu)) ) +

      .5*sum( (tu,m,mp),
      (u(m,tu) -utilde(m,tu))*lambda(m,mp)*(u(mp,tu) - utilde(mp,tu)));

stateq(n,t+1)..

x(n,t+1) =e=
      sum(np, (a(n,np)*x(np,t))) +
      sum(m, (b(n,m)*u(m,t))) +
      sum(k,(c(n,k)*z(k,t)));



Model track /all/;

x.fx(n,ti) = xinit(n);

Solve track minimizing j using nlp;


Display x.l, u.l;
```

Chapter 7

Portfolio Model in MATLAB

The classic portfolio optimization problem, which was originally proposed by Markowitz (1952), was to consider both the mean and the variance of a portfolio by maximizing the mean while minimizing the variance. This was formulated as a quadratic programming problem to maximize a weighted sum of the mean and the negative of the variance. Thus one could consider the tradeoff between stocks with high means and greater risk with their higher variances and stocks with low means and low risk with lower variances. Also, one could consider building a diversified portfolio which contained stocks that tended to move in opposite directions as represented by negative covariance elements.

Our goal in this chapter is to use MATLAB to solve the optimal portfolio problem. First, we will solve the problem using a simple Monte Carlo optimization search program. This will be useful to provide a simple introduction to the MATLAB programming language and at the same time to learn a little about a random search procedure for optimization.

Then, we will move on to solve the portfolio optimization problem using a MATLAB gradient optimization function. However, this code makes use of the Optimization Toolbox and not all users of MATLAB have this Toolbox available to them. Therefore, in Appendix 7C we provide a GAMS version of the same problem. Also, for some readers the GAMS version may be somewhat easier to understand and it can thereby serve them as a useful entry ramp to the MATLAB gradient optimization program.[11]

1. The Mathematics

Consider a vector whose elements are the fractions of the portfolio which is invested in each of the equities, i.e.

---

[11] Also some readers may want to solve models of higher dimension than those used in this chapter with modified versions of both the MATLAB code and the GAMS code in order to compare the computational speeds of the two software systems.

(1) $$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

where

x_i = the fraction of the portfolio invested in equity i

for an example portfolio with three equities.

Also there is a vector μ that contains the mean return on each of the equities, i.e.

(2) $$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \end{bmatrix} = \begin{bmatrix} 8 \\ 12 \\ 15 \end{bmatrix}$$

where

μ_i = the mean return on equity i

Notice in this example that the second and third equities have the highest mean returns of 12 and 15, respectively. These data for the means in Eq. (2) and the covariances shown below are for illustrative purposes and do not represent the return on particular equities or groups of equities.

We can then use the inner product of these two vectors, i.e.

(3) $$\mu' x = \begin{bmatrix} \mu_1 & \mu_2 & \mu_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

to obtain the mean return for the portfolio.

The variance for the portfolio is given in the covariance matrix Σ, that is

(4) $$\Sigma = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{bmatrix} = \begin{bmatrix} 6 & -5 & 4 \\ -5 & 17 & -11 \\ 4 & -11 & 24 \end{bmatrix}$$

where

σ_{ij} = the covariance of the returns on equities i and j

Notice in this example that the second and third equities, which have the highest mean returns, also have the highest variances of 17 and 24 respectively. Also, note that the off-diagonal elements in the covariance matrix have different signs. Thus, for example, when the return on the first equity falls, the return on the second equity tends to rise since the covariance is -5. Thus holding these two equities in the same portfolio provides a cushion when the return on the first equity declines and the return on the second equity rises.

The variance of the portfolio can then be written as

$$x'\Sigma x = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{5}$$

The Markowitz model considers both the mean and the variance of a portfolio by maximizing the mean while minimizing the variance. Using the components of the mean and variance of the portfolio from Eqs. (3) and (5) one can write the criterion function for the model as to maximize J in

$$J = \mu' x - \frac{1}{2}\beta x'\Sigma x \tag{6}$$

where

J = criterion value

β = subjective weight on the variance of the return on the portfolio

The parameter β provides the subjective weight on the variance. Thus an individual with a high β is risk averse and will choose a portfolio with equities which have relatively small variances. The one-half in this expression is commonly used in quadratic criterion functions; however it plays no essential role.

The constraint for this model simply requires that the fractions invested in each of the equities add to one, i.e.

$$\sum_{i \in I} x_i = 1 \tag{7}$$

where

I = the set of equities

Also there is a constraint which requires that the factions be nonnegative, i.e.

(8) $$x_i \geq 0 \quad i \in I$$

So, in summary, the model is to find those values of x_i that will maximize J in Eq. (6) subject to the constraints in Eqs. (7) and (8).

The optimal portfolio model can also be posed in a related way that seeks to find the fractional equity holdings that will minimize the weighted risk subject to a constraint that the mean return on the portfolio should be above a specified level. The criterion function for this model is

(9) $$J = \frac{1}{2}\beta y' \Sigma y$$

where

y = vector of fractions of portfolio invested in each equity

subject to

(10) $$\mu' y \geq \theta$$

where

θ = desired minimum mean-return on portfolio

(11) $$\sum_{i \in I} y_i = 1$$

(12) $$y_i \geq 0 \quad i \in I$$

In summary, this second version of the model is to find those values of y_i that will minimize J in Eq. (9) subject to the constraints in Eqs. (10) thru (12). The key parameter in this formulation is θ, the desired minimum mean-return for the portfolio. As this parameter is increased the optimal portfolio will include more of the risky equities.

This completes the statement of the mathematics of the two versions of the models. Next we turn to the computational statement of the models in MATLAB.

2. A Simple Monte Carlo Optimization Procedure in MATLAB

In order to provide a good opportunity to learn both the basics of the MATLAB software and the basics of a Monte Carlo optimization search procedure we have chosen a simple application to solve the first formulation of the Markowitz model. This application is based on some programs developed by our students Paul Maksymonko, Kevin Kline, and Carter Hemphill.[12] The optimization procedure includes the generation of a population of eight candidate portfolios in the first period. The portfolio that performs best is then selected and stored. Then the next period portfolios are generated as random variations around that portfolio. This process is repeated 100 times.

The basic structure of the program is a set of two `for` loops. The outside loop is across time periods (or "runs") and the inside loop is over candidates. These loops look something like the following

```
nruns = 100 ;  popsize = 8;
for k = 1:nruns;
      for i = 1:popsize;
            …
      end
      …
end
```

Notice that the indentation in the code above makes it easy to see the beginning and ending of each of the "for" loops. The indentation is not necessary for the MATLAB compiler but can (and should) be used to make the code easier to read.

The first step in the code is to initialize the number of time periods (or runs) and the number of candidates in each time period with the MATLAB statements

[12] They developed some applications to be used as an introduction to an optimization method known as Genetic Algorithms, a method particularly useful when dealing with non-convex problems. We will not deal with that method here, though the application we will present has a resemblance to that method. The approach used here is more like an evolutionary algorithm (EA) in that it uses real numbers rather than the strings of bits in that are used in many genetic algorithms (GA). Later in the book we will provide an introduction to genetic algorithms in a chapter on that subject.

```
nruns = 100;    popsize = 8;
```

This is later followed by the main `for` loop in the program which is over the time periods (or runs). The structure of the `for` loop is

```
for k =1:nruns;
      main body of the program
end
```

So the time index in this model is `k` and it runs from 1 to the number of runs. Also note that each `for` loop in MATLAB extends until the matching `end` statement is encountered. Thus it is useful when reading MATLAB code of this type to examine the structure of the code by looking for matching `for` and `end` statements. This is shown below in a pseudo code outline of the structure of the program. The code is called "pseudo" because it could not be run on a computer as it is but rather is intended to outline the basic structure of the program.

```
nruns = 100 ;  popsize = 8;
initialize portfolio weights

for k = 1:nruns;
      generate returns, variance costs and criterion values
      select best portfolio

      for i = 1:popsize;
            generate new random portfolio weights (percentages)
            for each candidate
      end

end
print and graph the sequence of best candidates
```

After the number of runs and the population size are set the initialization section of the code is used to set the initial portfolio weights for each candidate.

Then the `k` loop for the number of runs begins with the computation for that time period of the returns and the variance costs for each of the portfolios. These values are then used to calculate a vector which gives the criterion value which was obtained for each of the eight candidates. This criterion vector is then examined to find the index of the candidate with the highest criterion value. This

best portfolio is then used in the second, nested, `i` loop for the candidates the basis for the generation of the portfolio holding of the eight candidates in the next period.

After the time period (or run) loop is repeated 100 times the sequence of best portfolios in each period is printed and plotted.

With this overview of the program in mind consider next each of the sections of the code.

3. Initialization of Counters, Parameters and Weights

This section of the code contains the initialization of the counters for the number of runs (`nruns`), the population size (`popsize`) and the parameters of the portfolio model (the risk aversion coefficient (`beta`), the vector of mean returns (`mu`) and the covariance matrix (`sigma`)).

```
nruns = 100;  popsize = 8;
beta = 2;
mu = [8 12 15]';
sigma = [6 -5  4
        -5 17 -11
         4 -11 24];
```

There is also a constant

```
const = 0.1;
```

which will be used later to determine de degree of random variation around the weights of the best candidate of a time period to generate the candidates of the next period.

Finally, we create the vector of initial portfolios for the first time period with the statement

```
pwm = (1/3) * ones(3,popsize);
```

Thus `pwm` stands for the portfolio weight matrix. The function `ones()` generates a matrix of ones with three rows and a number of columns equal to the population size. Thus, `pwm` will contain eight column vectors with one portfolio each, all

with weights set equal to 1/3. Thus the initial `pwm` looks like

$$pwm = \begin{bmatrix} .33 & .33 & .33 & .33 & .33 & .33 & .33 & .33 \\ .33 & .33 & .33 & .33 & .33 & .33 & .33 & .33 \\ .33 & .33 & .33 & .33 & .33 & .33 & .33 & .33 \end{bmatrix}$$

Notice here that it is not necessary in MATLAB to first *declare* a variable and then *define* it. Declarations are used in many program languages to determine the type of a variable, viz. whether it is an integer or a floating point number and whether it is a scalar or a multidimensional array. Also, the declaration is used to set aside enough space in memory to store the elements of the variable before the numerical values of each element are defined in a separate statement in the language. Thus, in the statement

```
mu = [8 12 15]';
```

which is used above, the variable `mu` is both declared and defined by its context to be a column vector with three elements. The vector is input as a row vector but the transpose (`'`) mark is used to convert it to a column vector.

The next step is the generation of the returns, variance cost and criterion value for each portfolio.

4. Generation of Returns, Variance Costs and Criterion Values

The returns for every candidate are generated with the statement

```
pret = pwm' * mu;
```

where `pret` is an 8 element vector that contains the portfolio return for each of the eight candidates. The original `pwm` matrix is 3 x 8 as we saw above; therefore, its transpose which is used in the statement above is 8 x 3. This matrix in turn is multiplied by the 3 element column vector `mu` to yield the 8 element column vector `pret`.

Generating the variance costs for every candidate requires the use of a short loop

```
for j = 1:popsize;
      pvar(j) = 0.5 * beta * pwm(:,j)' * sigma * pwm(:,j);
end
```

The notation `(:,j)` in the matrix `pwm` refers to all the elements of the jth column of the matrix. Remember that each column in `pwm` corresponds to one portfolio. Thus by the time the code has passed through this loop eight times the variance costs for all the candidates are neatly stored in the `pvar` vector which has eight elements (one for each candidate).

The criterion values for each candidate are just the difference between the portfolios returns and the variance costs and are computed with the statement

```
pcrit = pret - pvar';
```

The vectors `pret` and `pvar` each have eight elements. Since the vector `pvar` is a row vector it has to be transposed in the expression above. Thus from the expression above pcrit is an 8 element column vector with the criterion value for each of the portfolios. Thus this vector can be used to find the best candidate. Of course in the first pass through this part of the code all portfolios are the same so the criterion values will be the same for all of them.

.

5. Selection of the Best Portfolio

The next step is to find the portfolio which has the highest rate of return. This is done with the statement

```
[top topi] = max(pcrit);
```

that uses the MATLAB function `max` to place in the scalar `top` the largest element in the vector `pcrit` and the corresponding index in the scalar `topi`. If there is more than one maximum, this function will choose only one.

The index is then used to put into the vector `wnew` the set of portfolio weights used by this candidate with the statement

```
wnew = pwm(:,topi);
```

Recall that the matrix `pwm` has *three* rows (one for each asset class) and eight columns (one for each candidate) so the effect of the statement above is to put the three elements from the `topi` column of the matrix into the vector `wnew`. The portfolio weights for the best candidate and the criterion value in each time period `k` are then stored in the matrices `wbest` and `pcritvec` using the statements

```
wbest(:,k) = wnew;
pcritvec(:,k) = top;
```

These arrays can then be used at the bottom of the code to plot the best portfolio in each run and the corresponding criterion value.

6. Random Generation of New Portfolios

The candidates for the next period are created as random variations around the portfolio weights of the best candidate from the previous period. The weights from the best candidate have been stored in the vector `wnew` and that vector is used in the `for` loop below to create eight new candidates.

```
for i = 1:popsize-1;
      w1 = wnew(1) + rand * const;
      w2 = wnew(2) + rand * const;
      w3 = wnew(3) + rand * const;
      temp = w1 + w2 + w3;
      w1 = w1/temp;
      w2 = w2/temp;
      w3 = w3/temp;
      pwnew(:,i) = [w1 w2 w3]';
end
```

The MATLAB random number generator, `rand`, for uniform distributions between zero and one is used here and is multiplied by a constant. This has the effect of adding a given amount to the portfolio weight for each equity. The weights are then normalized so they add up to one. The last statement in the loop above, i.e.

```
pwnew(:,j) = [w1;w2;w3];
```

simply stores the weight vector for the jth candidate in the jth column of the new portfolio weight matrix, `pwnew`. Thus by the time the loop has been completed the portfolio weights for the first seven candidates have been stored in the `pwnew` matrix.

The next step is to put the best portfolio from the previous run in the last (eighth) column of the `pwnew` matrix using the statement

```
pwnew(:,popsize) = wnew ;
```

This has the effect of keeping the best solution from each run when generating the new portfolios to be used for the next run.

Then the statement

```
pwm = pwnew
```

is used to replace the previous period matrix of portfolios by the newly generated matrix. Following this statement is the last `end` statement in the code. This is the `end` that corresponds to the `for` loop across time periods.

After the time period loop is completed the weights for the surviving candidate and the criterion values are printed with the simple statements

```
wnew
top
```

The absence of a semicolon at the end of these statements dictates that the result will be printed. Finally, the commands below generate a graph displaying the values of the *three* assets percentage holdings for the best candidate in each time period.

```
xaxis = [1:1:nruns]';
plot(xaxis,wsurv(:,:));
xlabel('Runs');
ylabel('Weights');
legend('w1', 'w2', 'w3');
```

Also we have commented out an additional statement that can be used to

plot the criterion value for all runs. It is

```
%plot(xaxis,pcritvec(:,:));
```

If you want to obtain this plot simply remove the leading % sign and rerun the program.

The entire code of the program is contained below in Appendix 7A and is also available in the book web site under the name `mcportfol.m`. The instructions for running MATLAB are in Appendix I.

It is important to point out that every time you run the program, particularly when changing the number of runs or the population size, you should clean out the old commands and workspace to avoid displaying spurious results. To do so, go to Edit in the top MATLAB menu. Then select Clear Command Window and confirm with Yes that you want to do this. Then do the same for Clear Command History and for Clear Workspace. Alternatively, adding the sentence

```
clear all;
```

at the beginning of the program will clear the workspace.

Figure 7.1 shows the sequence of weights of the best portfolios at each time period. The optimal portfolio weights for this experiment correspond to the last time period and are: w1 = 0.24, w2 = 0.43 and w3 = 0.33.

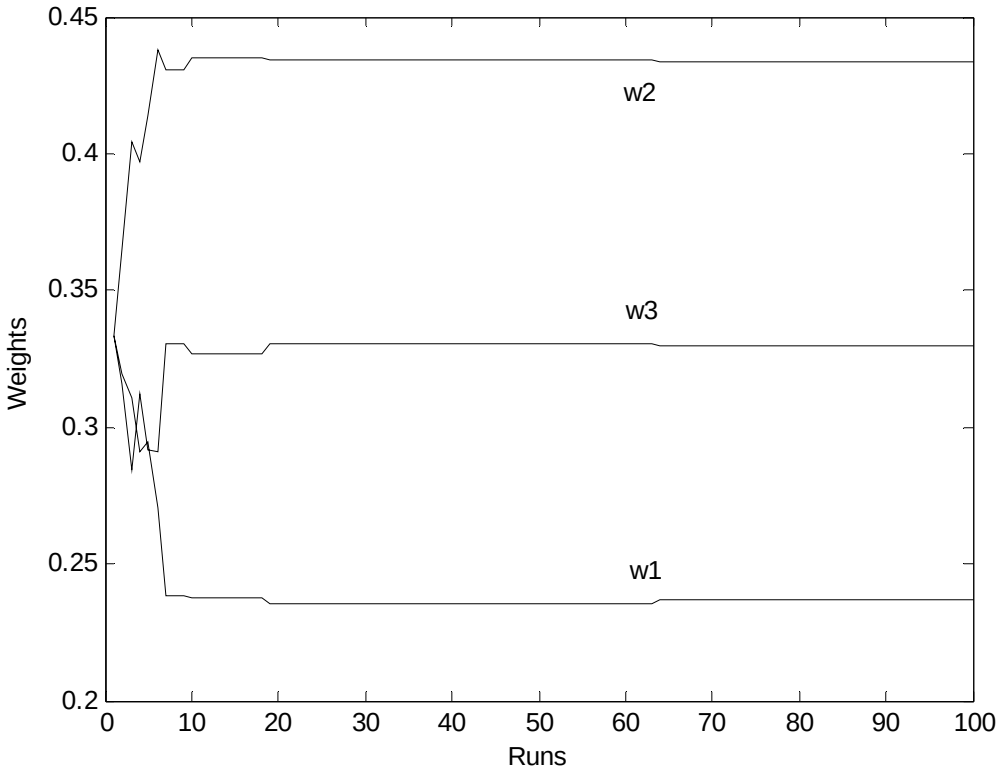


Figure 7.1 Best Portfolio at Each Time Period

Notice for this particular model and starting conditions that portfolios close to the optimum are found within only about ten runs.

This small random search optimization routine is simple to program and for the particular example problem is relatively effective in finding the optimal solution. More important it serves our purpose of introducing the MATLAB software with a relatively uncomplicated code that performs nicely on this simple problem.

However, to see the shortcoming of this simple code you can try solving the case where `beta` is set equal to zero. In this case the solution will be a boundary solution since the optimal portfolio will be one in which the entire portfolio is placed in the one equity with the highest mean return. The simple code above has a difficult time finding this solution but the more complex gradient method approach discussed in the next section finds that optimal solution with relative ease.

7. The Markowitz Model Using a MATLAB Optimization Function

We turn now to the solution of both versions of the Markowitz model using a MATLAB function from the Optimization Toolbox. Therefore, before beginning to work with this code, be sure that the version of MATLAB that you are using includes the Optimization Toolbox.

The function to be used is `fmincon`. It is designed to find the minimum of a function f(x) with linear inequality and equality constraints and with nonlinear constraints. Thus our model can be solved with a nonlinear optimization solver (see Appendix F). A simplified version of this function call for a model that has only linear inequality constraints would be (this function call is used only for exposition and will not necessarily work in a MATLAB program)

```
[x,fval] = fmincon(@func,x0,A,b)
```

where

```
x = the vector of optimal values
fval = the value of the criterion function at the optimum
fmincon = the name of the function from the Optimization
       Toolbox
func = the name of the user supplied function that returns
       the criterion value for the function
x0 = a vector of starting values to be used in the search for
       the optimal value of the function
A = the matrix for the linear inequalities  Ax <= b
b = the vector for the linear inequalities Ax <= b
```

To use the `fmincon` function in this case the user would have to supply a function

```
func
```

that would return the value of the criterion function. Also the user should provide a vector `x0` of values that he or she thinks is close to the optimal value of the function. This starting point is used by the Optimization Toolbox function as a starting point in the search for the optimal value of the function. Also, the user must supply the `A` matrix and the `b` vector for the linear inequality

```
Ax <= b
```

that constrains the solution to the model.

A somewhat more complicated version of the call to `fmincon` would include in addition to the linear inequalities also linear equalities and upper and lower bounds on the variables and would be of the form

```
[x,fval]=fmincon(@func,x0,A,b,Aeq,beq,lb,ub);
```

where

```
Aeq = the matrix for the linear equalities Aeq x = beq
beq = the vector for the linear equalities Aeq x = beq
lb = lower bound on the variables, i.e. lb <= x
ub = upper bound on the variables, i.e. x <= up
```

The actual call to `fmincon` is still more complicated in that it permits options to specify nonlinear constraints and to pass the model parameters to the criterion function. For the first version of the optimal portfolio model this function call is

```
[x,fval,exitflag,output]
         =fmincon(@dcri1,x0,A1,b1,Aeq,beq,lb,ub,nonlcon,options,beta,
             N,mu,sigma);
```

where

```
exitflag = provides info on the condition of the function at exit
output = provides additional output from the function
dcri1 = the user supplied function that returns
        the criterion value for the first version
        of the model
x0 = a vector of starting values to be used in the search for
        the optimal value of the function
A1 = the matrix for the linear inequalities  A1 x <= b1
        for the first version of the model
b1 = the vector for the linear inequalities A1 x <= b1
        for the first version of the model
nonlcon = specification for the nonlinear constraints
options = options to pass to the function
beta,N,mu,sigma = additional arguments to be passed to the function
```

For the second version of the model, where we minimize the variance subject to a constraint on the portfolio return, the call to the `fmincon` function is identical to the one above except that the user supplied function is named `dcri2` and that the matrix and vector for the set of linear inequalities are designated respectively as `A2` and `b2`.

The MATLAB code for the optimal portfolio model, which was programmed by Miwa Hattori, is shown in Appendix 7B and is also available in the book web site under the name `portfolio.m`. Other than the call to the function `fmincon`, the rest of the code is devoted primarily to preparing the inputs to pass to the function and to providing the function which returns the criterion value. So lets begin with the code to pass the parameters θ and β and the number of equities in the portfolio, `N`. This is written

```
theta=10;
beta=2;
N=3;
```

The next section of the code is used to input the values of the mean-return vector μ and the covariance matrix Σ and is

```
mu=[8; 12; 15];
sigma=[6  -5   4;
      -5  17 -11;
       4 -11  24];
```

Notice that each line of the vector `mu` is ended with a semicolon, so `mu` is input as a column vector.

The next step is to provide the starting values that are to be used in the search for the optimum shares in the portfolio. A reasonable starting point is to divide the portfolio equally among the three equities. This is accomplished in the MATLAB code with the statements

```
x0=ones(N,1)/N;
y0=ones(N,1)/N;
```

where

```
x0 = the vector of starting values to be used in the search for
       the optimal value of the function in the first version
       of the model
y0 = the vector of starting values to be used in the search for
       the optimal value of the function in the second version
       of the model
```

The MATLAB function `ones(m,n)` is used to create an `n` by `m` matrix of ones. So in this case the function call `ones(N,1)` creates an `N` vector of ones. All of the elements of this vector are then divided by `N`, so in our case with three equities the vector `x0` will have three elements all of which are 0.33. Also the same will be true for `y0` which is used with the second version of the model.

Next consider the linear inequality constraints for the two versions of the model. The first version has only a linear *equality* constraint and no linear inequality constraints so this is input with the MATLAB statements

```
A1=[];
b1=[];
```

i.e. the matrix `A1` and the vector `b1` are empty and can be ignored by the function. However this is not the case in the second version of the model which minimizes the weighted variance subject to achieving at least a minimum mean return on the portfolio, i.e.

(10) $$\mu' y \geq \theta$$

However, since MATLAB expects the inequality in less-than-or-equal form it is necessary to multiply the constraint through by minus one to obtain

(13) $$-\mu' y \leq -\theta$$

Then the `A2` matrix and the `b2` vector for this constraint can be input to the code with the statements

```
A2 = -mu';
b2 = -theta;
```

Since the vector `mu` was input above as a column vector it must be transposed with the transpose operator (') here since we need it in the form of a row vector for this constraint.

The equality constraints for the two versions of the model are the same and are of the form

$$\sum_{i \in I} x_i = 1$$

for the first version and

$$\sum_{i \in I} y_i = 1$$

for the second version. So the `A` matrix and `b` vector have the same structure for both versions of the model and can be input with the statements

```
Aeq = [1 1 1];
Beq = 1;
```

The lower bounds on the variables are used to enforce the non-negativity constraints and there are no upper bounds so the bounds for both versions of the model are specified with the statements.

```
lb=[0;0;0];
ub=[];
```

The final part of the model specification is the nonlinear constraints, of which there are none, so this is written

```
nonlcon=[];
options = optimset('MaxIter',60);
```

Also, the `options` variable is used to set the maximum number of iterations for the nonlinear programming code to 60. If the code has difficulty converging on the solution to your model it would be useful to raise this limit.

With all this preparation done, one can now call the `fmincon` function for the first versions of the model and print the key results with the statements

```
[x,fval,exitflag,output]
  = fmincon(@dcri1,x0,A1,b1,Aeq,beq,lb,ub,nonlcon,options,beta,
     N,mu,sigma);
x
fval
```

This MATLAB function will in turn call the user specified `dcri1` function for the first version of the model to obtain the value of the criterion function at each point x in the search for the optimum. Recall that for the first version of the model the criterion function in matrix form is, from Eq. (6) above

$$J = \mu' x - \frac{1}{2}\beta x' \Sigma x \tag{6}$$

which can be written in index form as

$$J = \sum_{i \in I} \mu_i x_i - \frac{1}{2}\beta \sum_{i \in I} \sum_{j \in J} x_i \sigma_{ij} x_j \tag{14}$$

This can be rearranged slightly by moving the β and x_i to obtain

$$J = \sum_{i \in I} \mu_i x_i - \frac{1}{2} \sum_{i \in I} x_i \left(\beta \sum_{j \in J} \sigma_{ij} x_j \right) \tag{15}$$

which is the form used in the `dcri1` function below.

```
function [z] = dcri1(x,beta,N,mu,sigma)
z=0;
for i=1:N;
        temp=0;
        for j=1:N;
              temp=temp+BETA*sigma(i,j)*x(j);
        end;
        z=z+mu(i)*x(i)-0.5*x(i)*temp;
end;
z=-z;
```

Notice at the top of the function that the `fmincon` function passes to the `dcri1` function the current point `x` in the search for the optimum and the parameter of the problem.

Notice at the bottom of the function that the negative value of `z` is returned by the function. The reason is that the `fmincon` function – as its name indicates – is used to find the minimum value of a function. Therefore to use it to find the maximum, as we need here, it is necessary to reverse the sign of the criterion value.

An equivalent but more compact formulation that shows the power of MATLAB for matrix computation could be

```
function z = dcri1(x,beta,N,mu,sigma) ;
z = -(mu'*x - 0.5*beta*x'*sigma*x);
```

The call to the `fmincon` function for the second version of the model, followed by the command lines to print the results, is

```
[y,fval,exitflag,output]
    =fmincon(@dcri2,y0,A2,b2,Aeq,beq,lb,ub,nonlcon,options,beta,
      N,mu,sigma);
y
fval
```

The `dcri2` function for the second version of the model is similar to the `dcri1` function except simpler since it does not contain the `mu` parameters.

$$J = \frac{1}{2}\beta y'\Sigma y$$

Also it is not necessary to use the negative sign at the bottom of the function since we are seeking a minimum in this case.

8. Experiments

The logical experiment to do with the Markowitz model is to change the β risk preference parameter to see how the optimal portfolio changes. As β increases one would expect the optimal portfolio to contain larger proportions of stocks with lower variances and – most likely – with lower mean returns.

Another useful experiment is to change the pattern of the signs of the off-diagonal elements in the Σ matrix. In the original version used in this chapter there is a mixture of positive and negative off diagonal elements. It would make interesting experiments to selectively change the signs of these elements and observe the results.

Finally, another useful experiment is to compare the outcomes of the Monte Carlo code against the ones obtained with the optimization function. You may want to increase the number of time periods and/or the population size, or change the value of the constant `const`, and see how these changes affects the outcome of the Monte Carlo code. Also, in the "random generation of new portfolios" section, you may want to divide the constant `const` by the number of runs index `k` and see how this affects the convergence path of the best weights to the optimal portfolio.

Of course the reader may want to obtain data on a set of stocks and bonds which are of particular interest to him or her and thus develop a personal version of the optimal portfolio model.

9. Further Reading

For a variety of financial models in MATLAB see Brandimarte (2001).

Appendix 7A

MATLAB Code for a Monte Carlo Portfolio Problem

```
%Monte Carlo portfolio program;
%Program name: mcportfol.m
%Developed by Ruben Mercado with modifications by Scott Schwaitzberg
%and David Kendrick

clear all;

%initialization of counters, parameters and weights;
nruns = 100; popsize = 8;
beta = 2;
mu = [8 12 15]';
sigma = [6  -5   4
        -5  17 -11
         4 -11  24];
const = 0.1;
pwm = (1/3) * ones(3,popsize);


for k = 1:nruns;
    % generation of vectors of returns, variance cost and crit function
    pret = pwm' * mu;
    for j = 1:popsize;
        pvar(j) = 0.5 * beta * pwm(:,j)' * sigma * pwm(:,j);
    end
    pcrit = pret - pvar';

    % selection of the best portfolio;
    [top topi] = max(pcrit);
    wnew = pwm(:,topi);
```

```
 % store best portfolio and the optimal criterion value for each run
    wbest(:,k) = wnew;
    pcritvec(:,k) = top;

    % random generation of popsize minus one new portfolios;
    for i = 1:popsize-1;
        w1 = wnew(1) + rand * const;
        w2 = wnew(2) + rand * const;
        w3 = wnew(3) + rand * const;
        temp = w1 + w2 + w3;
        w1 = w1/temp;
        w2 = w2/temp;
        w3 = w3/temp;
        pwnew(:,i) = [w1 w2 w3]';
    end

    % put best portfolio for the run in the last column of the matrix
    pwnew(:,popsize) = wnew ;
    pwm = pwnew;
end

%print optimal weights and optimal criterion value
wnew
top

%print and graph optimal weights and criterion value
%wbest
xaxis = [1:1:nruns]';
plot(xaxis,wbest(:,:));
xlabel('Runs');
ylabel('Weights');
legend('w1','w2','w3');
%plot(xaxis,pcritvec(:,:));
```

Appendix 7B

MATLAB Code for a Markowitz Optimal Portfolio Problem

```
% Title: Quadratic-Linear Programming for Mean Variance Portfolio
% Analysis
% Program name: portfolio.m
% by Miwa Hattori
%   Implementation of the mean-variance portfolio selection models
%   with two alternative formulations in Matlab:
%   (1) maximizing expected mean return, net of variance costs and
%   (2) minimizing the overall variance costs of portfolio.
clear all;
%
%   Preliminaries
%
theta=10;     %   Minimum mean-return on portfolio under formulation 2.
beta=2;       %   Subjective weight on returns variance of equities.
N=3;          %   Number of available equity types.
mu=[8; 12; 15];  %   Column vector of mean annual returns on equities 1
                 %   through N (%).
sigma=[6  -5   4;%   Table of covariances between returns on equities.
      -5  17 -11;
       4 -11  24];

%
%   Provide initial "guesses" for portfolio vectors.
%
x0=ones(N,1)/N; %   Column vector of fractions of portfolio invested in
                %   equity i, initialized to 1/N.
y0=ones(N,1)/N; %   Column vector of fractions of portfolio invested in
                %   equity i, initialized to 1/N.

%
%   Constraints for optimization
%   Matlab only has a function that solves a constrained nonlinear
%    MINIMIZATION problem. See Help file for function "fmincon".
%    fmincon finds a minimum of a multivariable function f(x)subject to
%             A*x<= b, Aeq*x= beq, lb<= x <=ub where x, b, beq, lb, and
%             ub are vectors, A and Aeq are murices.
%

A1=[];     %  Set of linear inequality constraints under formulation 1.
b1=[];
A2=-mu';   %  Set of linear inequality constraints under formulation 2:
```

```
b2=-theta; %   Desired minimum mean-return on portfolio y >= theta (%).
Aeq=[1 1 1];   %   Set of linear equality constraints.
beq=1;    % Fractions x(i) must add to 1, fractions y(i) must add to 1.
lb=[0;0;0]; %   Non negativity constraints on x(i) and y(i)
ub=[];
nonlcon=[];         %   Non linear constraints-- none in this problem.
options=optimset('MaxIter',60);

%
%   Definition of the criterion functions
%       Functions dcri1, dcri2 are called.  See files dcri1.m, dcri2.m.
%
[x,fval,exitflag,output]=fmincon(@dcri1,x0,A1,b1,Aeq,beq,lb,ub,nonlcon,
options,beta,N,mu,sigma);
x
fval
[y,fval,exitflag,output]=fmincon(@dcri2,y0,A2,b2,Aeq,beq,lb,ub,nonlcon,
options,beta,N,mu,sigma);
y
fval


% Title: Quadratic-Linear Progr for Mean Variance Portfolio Analysis
% Function Name: dcri1.m
% by Miwa Hattori
% The first formulation of the crit function for mean-variance port
% selection model.
% Defines the expected mean return, net of variance costs, which is
% to be maximized.

function z = dcri1(x,beta,N,mu,sigma);

z=0;
for i=1:N;
    temp=0;
    for j=1:N;
        temp = temp + beta*sigma(i,j)*x(j);
    end;
    z = z + mu(i)*x(i) - 0.5*x(i)*temp;
end;

z=-z;

% Matlab only has a subrou to solve constrained MINIMIZATION problems.
% We solve a maximization problem by minimizing the negative of the
% objective function.
```

```
% Title: Quadratic-Linear Programming for Mean Variance Portfolio
% Analysis
% Function Name: dcri2.m
% by Miwa Hattori
% The second formulation of the criterion function for mean-variance
% portfolio selection model.
% Defines the overall variance costs of portfolio to be minimized.

function z = dcri2(y,beta,N,mu,sigma);

z=0;
for i=1:N;
    temp=0;
    for j=1:N;
        temp = temp + beta*sigma(i,j)*y(j);
    end;
    z = z + 0.5*y(i)*temp;
end;
```

Appendix 7C

GAMS Code for a Markowitz Optimal Portfolio Problem

The complete GAMS version of the model, which was programmed by Seung-Rae Kim, is at the end of this Appendix. Here we will discuss the parts of the model. The first part of the GAMS statement of the model is the specification for the set of equities

```
Set i   equities   /equity1, equity2, equity3/;
```

While it is more common in GAMS to use an upper case letter for the set so that the specification would be "Set I" instead of "Set i" there is an argument for using the lower case specification as is done here. The argument is that in GAMS the symbols for sets are used where the mathematics of the model would indicate a set *and* where the mathematics would indicate an index. Thus the mathematical statement of Eq. (7), i.e.

$$\sum_{i \in I} x_i = 1 \tag{7}$$

is written in GAMS as

```
sum(i, x(i)) =e= 1.0 ;
```

Of course, since GAMS does not distinguish between upper and lower case letters it would be possible to write the GAMS statement as

```
sum(I, x(i)) =e= 1.0 ;
```

This might be more aesthetically pleasing but could also be more confusing.

Just beneath the set specification statement in GAMS is an `Alias` statement of the form

```
Alias  (i,j) ;
```

This statement creates a set J which is a copy of the set I. This kind of statement is used in GAMS when there is a double summation over the elements of a variable `x(i,j)` of the sort that is used in computing the variance of the portfolio in this model.

Next the data are input using the `Scalar` keyword for θ and β, the `Parameters` keyword for the vector μ and the `Table` keyword for the matrix Σ as follows:

```
Scalar  theta desired minimum mean-return on portfolio (%) / 10 /
        beta  subjective weight on returns variance of equities / 2 /;

 Parameters  mu(i) mean annual returns on equities (%)
       / equity1    8
         equity2   12
         equity3   15 / ;

 Table sigma(i,j)  covariance matrix of returns on equities
                   equity1   equity2   equity3
         equity1      6        -5          4
         equity2     -5        17        -11
         equity3      4       -11         24 ;
```

Then the variables are defined using the keyword "`Variables`"

```
Variables
    x(i) fraction of portfolio invested in equity i in formulation 1
    y(i) fraction of portfolio invested in equity i in formulation 2
    criterion1  expected mean return on portfolio, net of variance cost
    criterion2  variance-augmented total risk cost of portfolio ;

Positive Variable x, y ;
```

Also, the `Positive Variable` statement is used in GAMS to enforce the non-negativity constraints on the `x` and `y` variables. Notice that the two versions of the model using the `x` variables in the one version and the `y` variables in the other are being developed simultaneously in the GAMS statement of the models, rather than one after another.

Next comes the declaration of the equations with the statements

```
Equations   dcri1   definition of criterion1
            dcri2   definition of criterion2
            xsum    fractions x(i) must add to 1.0
            dmu     desired minimum mean-return on portfolio y(i)
            ysum    fractions y(i) must add to 1.0 ;
```

Recall that the semicolon after the last line above is crucial in GAMS. It is easy to forget this semicolon when developing a model in GAMS; however, forgetting it usually results in errors in GAMS, since the compiler does not know where the list of equation names ends and the definition of the equations begins.

Next are the definitions of the equations beginning with the criterion function. However, since GAMS uses index rather than matrix notation, it is useful to restate the matrix form of the criterion function from Eq. (6) above, i.e.

$$J = \mu' x - \frac{1}{2}\beta x' \Sigma x \tag{6}$$

in index form as

$$J = \sum_{i \in I} \mu_i x_i - \frac{1}{2}\beta \sum_{i \in I} \sum_{j \in J} x_i \sigma_{ij} x_j \tag{14}$$

where

μ_i = the mean return on equity i

σ_{ij} = the covariance of the returns on equities i and j

This criterion is written in GAMS as

```
dcri1..  criterion1 =e= sum(i, mu(i)*x(i))
                        -.5*sum(i, x(i)*sum(j, beta*sigma(i,j)*x(j))) ;
```

Here we see the use of the `alias` I and J sets for the double summation. Similarly the criterion function for the second version of the model which is written in matrix form as

(9) $$J = \frac{1}{2}\beta y'\Sigma y$$

becomes

```
dcri2..  criterion2 =e= .5*sum(i, y(i)*sum(j, beta*sigma(i,j)*y(j))) ;
```

in the GAMS statement.

The rest of the constraints for the two versions of the model are stated in GAMS as

```
xsum..      sum(i, x(i)) =e= 1.0 ;
dmu..       sum(i, mu(i)* y(i)) =g= theta ;
ysum..      sum(i, y(i)) =e= 1.0 ;
```

The first and third constraints above require that the fractional portfolio holdings add to one. The middle constraint containing the θ parameter is the restriction on the portfolio return in the second version of the model.

The last part of the GAMS statement for the two versions of the model is

```
Model portfolio1  / dcri1, xsum / ;
Model portfolio2  / dcri2, dmu, ysum / ;

Solve portfolio1 using nlp maximizing criterion1;
Solve portfolio2 using nlp minimizing criterion2;
```

Here we see a good example in GAMS of the use of `Model` statements to specify different versions of a model that can then be solved one after another with two different `Solve` statements. The two models are actually quadratic programming models; however, GAMS does not have a specialized solver for this purpose and the nonlinear programming solver called by the keyword `nlp` is appropriate. For an introduction to this type of solver see Appendix F.

Below is the complete GAMS code for the Markowitz problem. The GAMS library has a variety of optimal portfolio models which may be of interest to the reader. They are called PORT and QP1 thru QP6. The PORT model was

created by the Control Data Corporation and the QP models were created by Erwin Kalvelagen at the GAMS Corporation.

```
$Title A Quadratic-Linear Program for Mean-Variance Portfolio Analysis
* Program by Seung-Rae Kim

$Ontext
These are mean-variance portfolio selection models with two
alternatives
formulations in GAMS: (1) maximizing expected mean return, net of
variance costs, & (2) minimizing the overall variance costs of
portfolio.
$Offtext

 Set i   equities   /equity1, equity2, equity3/;
 Alias   (i,j) ;

 Scalar  theta desired minimum mean-return on portfolio (%) / 10 /
         beta  subjective weight on returns variance of equities / 2 /;

 Parameters  mu(i) mean annual returns on equities (%)
       / equity1    8
         equity2   12
         equity3   15 / ;

 Table sigma(i,j)  covariance matrix of returns on equities
                   equity1   equity2   equity3
         equity1      6        -5          4
         equity2     -5        17        -11
         equity3      4       -11         24 ;

 Variables
    x(i) fraction of portfolio invested in equity i in formulation 1
    y(i) fraction of portfolio invested in equity i in formulation 2
    criterion1  expected mean return on portfolio, net of variance cost
    criterion2  variance-augmented total risk cost of portfolio ;

 Positive Variable x, y ;

 Equations  dcri1   definition of criterion1
            dcri2   definition of criterion2
            xsum    fractions x(i) must add to 1.0
            dmu     desired minimum mean-return on portfolio y(i)
            ysum    fractions y(i) must add to 1.0 ;
```

```
dcri1..  criterion1 =e= sum(i, mu(i)*x(i))-.5*sum(i, x(i)*sum(j,
                          beta*sigma(i,j)*x(j))) ;
dcri2..  criterion2 =e= .5*sum(i, y(i)*sum(j, beta*sigma(i,j)*y(j))) ;

xsum..     sum(i, x(i)) =e= 1.0 ;
dmu..      sum(i, mu(i)*y(i)) =g= theta ;
ysum..     sum(i, y(i)) =e= 1.0 ;

Model portfolio1  / dcri1, xsum / ;
Model portfolio2  / dcri2, dmu, ysum / ;

Solve portfolio1 using nlp maximizing criterion1;
Solve portfolio2 using nlp minimizing criterion2;
```

Part II

Once More . . .

Chapter 8

General Equilibrium Models in GAMS

The analysis of economy-wide models is a particularly demanding topic in economics, since it involves the study of interdependence. It implies a move to the realm of multiple heterogeneous agents, sectors and institutions interacting in complex ways. While there are some analytical methods and results available to help us in such endeavors, computational methods become necessary when we move to medium or large size models or when we deal with particularly complex ones.

This chapter provides an introduction to the art of economy-wide modeling. We present a sequence of small models, we show how to implement them in GAMS and we perform some experiments and suggest other experiments. We start with an Input-Output model in which quantities produced are determined given technology and demand levels. We follow with a Production Prices model that determines relative prices given technology and a distributive variable. Then we move to a General Equilibrium model in which prices and quantities are determined simultaneously given technology, preferences and endowments. Finally, we introduce SAM based and Johansen style Computable General Equilibrium models. We will present models in a sequence reflecting mainly their computational complexity in terms of degree of non-linearity and size. The order of the sequence does not mean historical or theoretical precedence of one type of model over the others, or a ranking of practical relevance.

1. Input-Output Model

A good starting point for the study of interdependence in economics is the well known Input-Output model pioneered by Nobel prize winner Wassily Leontief (1953). One of the main goals of this type of model is the determination of direct and indirect levels of production to satisfy a given increase in final demand.

Consider an economy with three industries (1, 2 and 3). Each of them produces a single output, using as inputs part of its own production as well as part of the output from

the other industries. It is clear, then, that each industry plays a dual role since it is both a supplier of inputs and a user of outputs. Imagine that each product in this economy is also used to satisfy an exogenously given level of demand from consumers. In formal terms, we can represent the economy just described as follows

$$\begin{aligned} x_1 &= a_{11}\ x_1 + a_{12}\ x_2 + a_{13}\ x_3 + d_1 \\ x_2 &= a_{21}\ x_1 + a_{22}\ x_2 + a_{23}\ x_3 + d_2 \\ x_3 &= a_{31}\ x_1 + a_{32}\ x_2 + a_{33}\ x_3 + d_3 \end{aligned} \tag{1}$$

where the x´s are production levels, the a_{ij} are the input-output coefficients (the intermediate requirements from industry *i* per unit of output of industry *j*), and the *d*'s are the levels of final demand from the consumers.[13] In matrix notation, we can write Eq. (1) as

$$x = Ax + d \tag{2}$$

where x is the vector of levels of production, d is the vector of final demands and A is the input-output coefficients matrix.

A question can be posed for this economy. Given an example input-output coefficients matrix

$$A = \begin{bmatrix} 0.3 & 0.2 & 0.2 \\ 0.1 & 0.4 & 0.5 \\ 0.4 & 0.1 & 0.2 \end{bmatrix}$$

and an example vector of final demands

$$d = \begin{bmatrix} 4 \\ 5 \\ 3 \end{bmatrix}$$

[13] One of the attractive features of input-output models is that in principle the data that is used to compute the coefficients in the model can be obtained directly from sources such as the manufacturing censuses done in many countries.

what will be the required level of total production of each industry (direct and indirect) to satisfy that final demand vector? The GAMS representation of this problem is

```
$TITLE IO-1
* Input-Output Model

SCALARS
d1 final demand for x1  /4/
d2 final demand for x2  /5/
d3 final demand for x3  /3/;

VARIABLES
x1 production level industry 1
x2 production level industry 2
x3 production level industry 3
j  performance index;

EQUATIONS
eqx1
eqx2
eqx3
jd performance index definition;

jd..    j =E= 0;
eqx1.. x1 =E= 0.3*x1 + 0.2*x2 + 0.2*x3 + d1;
eqx2.. x2 =E= 0.1*x1 + 0.4*x2 + 0.5*x3 + d2;
eqx3.. x3 =E= 0.4*x1 + 0.1*x2 + 0.2*x3 + d3;

MODEL IO /jd, eqx1, eqx2, eqx3/;
SOLVE IO MAXIMIZING J USING LP;
DISPLAY x1.l, x2.l, x3.l;
```

The GAMS files for this and the other models in this chapter are in the book web site under the names listed in each `$TITLE` statement. Note that in this model, as discussed in Appendix H "The Stacking Method in GAMS", in order to solve a system of simultaneous equations in GAMS it is necessary to add an additional variable (`j`) and equation (`jd`) and to maximize or minimize the added variable. As is discussed in that appendix, GAMS has procedures for optimizing but not for solving simultaneous equations. Therefore, the method for solving simultaneous equations in this software system is to add to the model an additional variable – `j` in this case – and an additional equation – `jd` in this case. Then the additional variable is maximized or minimized in order to find the solution to the model.

Using this method the solution obtained is

$$x_1 = 16.821,\ x_2 = 23.744,\ x_3 = 15.128.$$

There are analytical methods available to deal with this problem.[14] Indeed, the analytical solution is obtained by solving Eq. (2) to obtain

(3) $$x = \left(I - A\right)^{-1} d$$

where *I* is the identity matrix.[15] This formula can be easily handled for small models. However, computational methods will be required to perform the matrix inversion as soon as one moves to larger models. And these methods will become unavoidable as we move to more complex problems. For example, imagine now that we have some restriction, like a capacity constraint, on the maximum level of production of some products (say $x_2 \leq 22$ and $x_3 \leq 14$) and we want to know the maximum level of final demand of product 1 (d_1) that the economy can satisfy, given the final demand levels d_2 and d_3. This can be easily handled in GAMS. Here is the corresponding GAMS representation of the problem

```
$TITLE IO-2
* Input-Output Model with restrictions

SCALARS
d2 final demand for x2  /5/
d3 final demand for x3  /3/;

POSITIVE VARIABLES
x1 production level industry 1
x2 production level industry 2
x3 production level industry 3
d1 final demand for x1;

VARIABLES
j  performance index;
```

[14] See for example Chiang (1984) for an introduction to these methods.

[15] Also, it is necessary that the I-A matrix be non-singular.

```
EQUATIONS
eqx1
eqx2
eqx3
res1 restriction 1
res2 restriction 2
jd performance index definition;

jd..    j =E= d1;
eqx1.. x1 =E= 0.3*x1 + 0.2*x2 + 0.2*x3 + d1;
eqx2.. x2 =E= 0.1*x1 + 0.4*x2 + 0.5*x3 + d2;
eqx3.. x3 =E= 0.4*x1 + 0.1*x2 + 0.2*x3 + d3;
res1.. x2 =L= 22;
res2.. x3 =L= 14;

MODEL IO /all/;
SOLVE IO MAXIMIZING j USING LP;
DISPLAY x1.l, x2.l, x3.l, d1.l;
```

Notice that we define and add two equations (*res1* and *res2*) corresponding to the restrictions, set the performance index j equal to d_1, and define d_1 as a variable (no longer as a scalar). Also, to avoid negative values that make no economic sense we define all variables except the performance index as positive variables. Solving the problem, we obtain

$$x_1 = 14.143,\ x_2 = 22,\ x_3 = 13.571,\ d_1 = 2.786$$

in contrast with our original solution of

$$x_1 = 16.821,\ x_2 = 23.744,\ x_3 = 15.128,\ d_1 = 4$$

Thus the level of final demand for good 1 is lower once the restrictions are in place and we can achieve only 2.786. This is lower than in the original case since we set the values of the restrictions below the solution levels previously obtained. On the contrary, if the economy is able to lift those "bottlenecks" up to 30 for x_2 and 20 for x_3, the demand of goods produced by sector 1 that could be satisfied would be $d_1 = 7.8$.

2. Production Prices Model

So far we have been dealing with a model with two main types of agents (consumers and industries), in which their interrelations are linear and where, given a technology (the input-output coefficients matrix) we determine quantities produced and/or demanded. Implicitly, relative prices are taken as given. We will move now to a nonlinear model in which prices are determined given technology and a distributive variable. This type of model was pioneered by David Ricardo (1817) at the beginning of the nineteenth century and later formalized by Piero Sraffa (1972). One of its main goals is to allow us to study issues of income distribution between wages and profits.

Let's define

v = value of intermediate inputs
π = profits
w = wage cost
p = price.

We can then write

(4) $$v + \pi + w = p\,.$$

This equation simply requires that the total cost, that is the sum of the three elements of cost, namely intermediate goods, capital and labor is equal to the price. Then assuming that profits are equal to the profit rate r times the value of the intermediate inputs we have

(5) $$v + vr + w = p$$

or

(6) $$v(1+r) + w = p$$

Then using the input output coefficients for the intermediate inputs, a simple three-good production prices model can be formalized as

$$
\begin{aligned}
&(a_{11}\ p_1 + a_{21}\ p_2 + a_{31}\ p_3)\ (1+r) + l_1 w = p_1 \\
&(a_{12}\ p_1 + a_{22}\ p_2 + a_{32}\ p_3)\ (1+r) + l_2 w = p_2 \\
&(a_{13}\ p_1 + a_{23}\ p_2 + a_{33}\ p_3)\ (1+r) + l_3 w = p_3
\end{aligned} \tag{7}
$$

The *a*'s are, as before, input-output coefficients. Notice that subscripts of these coefficients are reversed, that is, the input-output matrix is the transpose of the *A* matrix corresponding to input-output Leontief type models. This is so because here we determine prices given technology, while in Leontief models we determine quantities given technology.[16] The *l*'s are also input-output coefficients indicating the quantity of labor required for the production of one unit of product. In addition the *p* 's are relative prices, *w* is the wage per unit of labor (assumed to be uniform for the whole economy) and *r* is the profit rate. The profit rate is the same for every industry, implying that we are dealing with a long run situation in which capital earns the same profit no matter the industry. Otherwise there would be capital movements from industries with a low rate to industries with a higher rate until that rate equalizes across industries.

The model above has five variables and three equations. Since all prices are relative prices, we need to choose one of them as a numeraire in order for all the other price-like variables to be expressed in terms of it. We can do this by fixing one variable (say, one price).[17] Once we have done this, to close the system of equations we are still left with a degree of freedom regarding *w* and *r*. We can thus fix, for example, the wage *w*.[18]

A GAMS representation of this model is provided below, where we have chosen a particular set of values for the input-output coefficients, and where we set $p_1 = 1$ and $w = 0$.

---

[16] To learn more about this, see Passinetti (1977).

[17] For Sraffa, the choice of the numeraire involved other issues dating back to Ricardo. Facing a change in the relative price of a commodity, Ricardo wanted to be able to tell when the change originated in the conditions affecting the production of that commodity or in the conditions of production of the commodity being used as numeraire. To solve in part that problem, Sraffa built a numeraire that takes the form of a restriction involving some of the model variables. This is a complex theoretical issue and we will not deal with it here. See Sraffa (1972).

[18] Classical economists like Ricardo used to consider that *w* was determined by the minimum subsistence level of the labor force. More modern approaches have considered that *w* was the outcome of the bargaining process between workers' unions and industrialists' unions.

```
$TITLE ProdPri
* Production Prices Model

SCALARS
L1 /0.2/
L2 /0.5/
L3 /0.3/;

VARIABLES
p1
p2
p3
w
r
j  performance index;

EQUATIONS
eqp1
eqp2
eqp3
jd performance index definition;

jd..    j =E= 0;
eqp1.. (0.3*p1 + 0.1*p2 + 0.4*p3) * (1+r) + L1 * w =E= p1;
eqp2.. (0.2*p1 + 0.4*p2 + 0.1*p3) * (1+r) + L2 * w =E= p2;
eqp3.. (0.2*p1 + 0.5*p2 + 0.2*p3) * (1+r) + L3 * w =E= p3;

w.fx = 0;
p1.fx = 1;
MODEL PP1 /all/;
SOLVE PP1 MAXIMIZING J USING NLP;
DISPLAY p1.l, p2.l, p3.l, w.l, r.l;
```

Notice that the statements

```
w.fx = 0;
p1.fx = 1;
```

are used to fix `w` and `p1`.

The solution for *r* is 0.25. It is interesting to observe what happens as we decrease *r*. To do so, we now set *r* equal to different fixed values, that is, we substitute `r.fx = 0.25` (and later `r.fx=0.20`, etc) for `w.fx = 0` in the GAMS representation above. We will find that there is an inverse relationship between the wage *w* and the profit rate *r*, such as the one shown in Table 8.1.

r	*w*
0.25	0.000
0.20	0.157
0.15	0.270
0.10	0.389
0.05	0.515
0.00	0.648

Table 8.1 Wages and Profits

In this example, not only wages, but also prices go up as *r* decreases. However, in general, prices can go either way - some may go up, others down. However, if we choose *w* as the numeraire, we will observe that as *r* increases, all prices increase, indicating that the real wage will decrease no matter the weights used to compute the corresponding wage deflator.

3. General Equilibrium Model

In the previous two sections we considered first a quantity model and then a price model. Here we move to a model in which quantities and prices are determined simultaneously. General equilibrium models of this type were pioneered by Leon Walras (1834-1910) (cf. Walras, L., *Elements of Pure Economics*, Augustus M. Kelley Publishers (1969)) and generalized by Nobel Prize winners Kenneth Arrow (Arrow and Hahn (1971)) and Gerard Debreu (1986). One of the main goals of general equilibrium modeling is the study of changes in prices and quantities when technology, preferences or endowments change.

Imagine that we have a very simple economy, with only one production sector, two factors of production and a single household. The production sector produces a single good q_s (output supply) with a Cobb-Douglas constant returns to scale production technology using two inputs: labor and capital. Technical progress (*b*) can affect total factor productivity. The corresponding labor and capital demand functions (l_d and k_d) are derived combining the production function with the assumption of profit maximizing behavior. Labor and capital supplies (l_s and k_s) are given exogenously. The single household provides labor and capital in exchange for the corresponding wage (*w*) and

profit (*r*), spending all its income (*y*) in the demand for the single good (q_d). So far, we have three markets: labor, capital and good markets, and we impose market clearing conditions specifying that supply equals demand. The model equations are listed below[19]

production function (Cobb-Douglas)

(8) $$q_s = b\ l_d^a\ k_d^{1-a}$$

labor demand, supply and market clearing

(9) $$l_d = \frac{a\ q_s\ p}{w}, \qquad l_s = \overline{l}_s, \qquad l_s = l_d$$

capital demand, supply and market clearing

---

[19] To obtain the expressions for the labor and capital demand functions, we maximize the profit function

$$\pi = p\,q_s - w\,l_d - r\,k_d$$

subject to the production function

$$q_s = b\ l_d^a\ k_d^{1-a}$$

Substituting the production function into the profit function, the first order conditions are

I) $$\frac{\partial \pi}{\partial l_d} = p\,a\,b\,l_d^{a-1}\,k_d^{1-a} - w = 0$$

II) $$\frac{\partial \pi}{\partial k_d} = p\ (1-a)\,b\,l_d^a\ k_d^{-a} - r = 0$$

Substituting the production function into I and II and rearranging terms we obtain, respectively, the labor and capital demand functions

$$l_d = \frac{a\,q_s\ p}{w} \quad \text{and} \quad k_d = \frac{(1-a)\,q_s\ p}{r}$$

(10) $$k_d = \frac{(1-a)\, q_s\, p}{r}, \qquad k_s = \bar{k}_s, \qquad k_s = k_d$$

household income

(11) $$y = w\, l_d + r\, k_d$$

good demand

(12) $$q_d = \frac{y}{p}$$

good market clearing

(13) $$q_s = q_d$$

This simple model has 10 variables and 10 equations. However, one of them is redundant, since "Walras law" establishes that for n-markets we need n-1 equilibrium conditions only. Also, since this model determines relative prices (p, w and r), we need to fix one of them as the numeraire. Thus, by choosing one price as the numeraire (say we fix $p = 1$) and deleting the corresponding good market clearing equation ($q_s = q_d$), we are left with a 9-variable 9-equation well-defined model. We do not consider the performance index `j` (that is used in the GAMS representation below) in the variable count nor the performance index definition in the equation count.

The GAMS representation of the model is shown below. Arbitrary, but reasonable, numbers have been chosen for the parameters and for the labor and capital stocks.

```
$TITLE SIMPLEGE

SCALARS
a labor share / 0.7 /
b technology parameter / 1.2 /;

POSITIVE VARIABLES
qs good supply
qd good demand
ld labor demand
ls labor supply
kd capital demand
ks capital supply
p price
w wage
r profit
y income;
```

```
VARIABLES
j  performance index;

EQUATIONS
eqs good supply equation (production function)
eqd good demand equation
eld labor demand equation
els labor supply equation
ekd capital demand equation
eks capital supply equation
ey  income equation
eml labor market clearing
emk capital market clearing
jd performance index definition;

jd..       j =E= 0;

eqs..     qs =E= b * ld**a * kd**(1-a);
eld..     ld =E= a * qs * p / w;
els..     ls =E= 2;
eml..     ld =E= ls;
ekd..     kd =E= (1-a)* qs * p / r;
eks..     ks =E= 1;
emk..     kd =E= ks;
ey..       y =E= w * ld + r * kd;
eqd..     qd =E= y / p;

*lower bounds to avoid division by zero
p.lo = 0.001;  w.lo = 0.001;  r.lo = 0.001;

*numeraire
p.fx = 1;

MODEL SIMPLEGE /all/;
SOLVE SIMPLEGE MAXIMIZING J USING NLP;
DISPLAY qs.l, qd.l, ld.l, ls.l, kd.l, ks.l, p.l, w.l, r.l, y.l;
```

The solution values are

```
qs.L  =   1.949  good supply
qd.L  =   1.949  good demand
ld.L  =   2.000  labor demand
ls.L  =   2.000  labor supply
kd.L  =   1.000  capital demand
ks.L  =   1.000  capital supply
p.L   =   1.000  price
w.L   =   0.682  wage
r.L   =   0.585  profit
y.L   =   1.949  income
```

It is important to perform some basic checks on the workings of the model. For instance, since we assumed market clearing, we have to verify that supply equals demand in each market. Also, when increasing the value of the numeraire, all quantity variables should remain the same, while nominal variables (prices and income) should increase proportionally. Notice that this model, as the other models previously introduced, are models of the "real" side of the economy, in the sense that money is not explicitly included in them. Also, the result that real variables remain the same while nominal variables change in proportion to the numeraire can be interpreted as meaning that money is neutral in this model.

4. Computable General Equilibrium Models

So far we have presented very small models. However, applied economy-wide models tend to be large, thus making the use of computational techniques unavoidable. In this section we will introduce a slightly larger model than the General Equilibrium model presented in section three, to have a flavor of what is like to deal with more than a handful of variables and equations. Models like this are known in the literature as Computable General Equilibrium (CGE) models. We will later go back to a small model to illustrate the application of a linearization technique useful when dealing with relatively large nonlinear models. The material in the remainder of this chapter is considerably more difficult than in the previous sections. Also, the exposition moves at a more rapid pace.

4.1 A SAM Based Model

We move now to a two-sector, two-factor and two-household model to illustrate how to build a CGE model based on a Social Accounting Matrix (SAM). This model was developed by Arne Drud at the World Bank and is discussed in Kendrick (1990).[20]

Following the research of Nobel prize winner Richard Stone (1961), a SAM contains information on the flow of goods and payments between institutions in the economy. In Table 8.2 we present a simple SAM where the table should be read following the principle that columns pay rows and where each column adds up to the same number as the corresponding row.

	Factors		**Households**		**Sectors**	
	Labor	Capital	Rural	Urban	Food	Clothing
Factors						
Labor					75	85
Capital					50	60
Households						
Rural	90	30				
Urban	70	80				
Sectors						
Food			60	65		
Clothing			60	85		

Table 8.2 A Simple SAM

[20] Drud implemented the model in Hercules, a system which allowed the modeler to develop CGE models by providing basic information in the form of Social Accounting Matrices and by choosing from a menu the functional forms for production functions and demand functions. Hercules is no longer in use; however GAMS now provides a solver (MPSGE) which performs similar functions to those of Hercules (see www.gams.com). These types of systems for model representation are very useful and especially time saving for the experienced modeler. However, here we will present a direct GAMS representation of the Drud model which is more suitable to introduce beginners to basic issues in computational model building.

For example, the food industry pays 75 to labor and 50 to capital. Labor pays 90 to rural households and 70 to urban households. Urban households spend 65 on food and 85 on clothing.

Usually, a SAM can be constructed using a country's official statistics such us the national accounts. Based on the table above, Drud built the model shown in Table 8.3.

	Quantity q	**Price, Share or Payment** p	**Price-Quantity** pq
Sectors			
Output	$q_s = b_s \prod_f c_{fs}^{a_{fs}}$		$y_s = p_s \; q_s$
Input	$c_{fs} = \frac{a_{fs} \; q_s \; p_s}{p_f}$		$t_{fs} = p_f \; c_{fs}$
Factors			
Income			$y_f = p_f \; q_f$
Transfer		$t_{hf} = a_{hf} \; q_f$	
Household			
Consumption		$t_{sh} = a_{sh} \; q_h$	$t_{sh} = p_s \; c_{sh}$
CPI		$p_h = \prod_s p_s^{a_{sh}}$	$y_h = p_h \; q_h$
Linkage			
Sectors		$y_s = \sum_h t_{sh}$	
Factors		$y_f = \sum_s t_{fs}$	
Households		$y_h = \sum_f t_{hf}$	

Table 8.3 Drud's Model

The model contains three key types of variables: price (*p*), quantity (*q*) and income (*y*), all of them with a single subscript since they apply to a single institution (subscript *f* indicates factor, *h* household and *s* sector). There are also two additional types of variables: payment (*t*) and commodity (*c*), with two subscripts since they represent flows of goods and payment. The subscripts on the payment variables t follow the SAM convention: payments are from columns to rows (i.e. t_{fs} indicates payment from sector *s* to factor *f*). Commodity flows c follow the more common forward subscript convention (i.e. c_{fs} indicates the flow of factor *f* to sector *s*, while c_{sh} is the flow of purchased goods from sector *s* to household *h*).

The output-quantity equations specify production functions with a Cobb-Douglas technology where *b* is a technology parameter. The input-quantity equations are the corresponding factor demand equations derived from the production functions and imposing a zero profit condition. The CPI-price equations are price indexes for the rural and urban households respectively. The *a ´*s are share parameters derived from the SAM.

When expanded, the model has 38 variables and 36 equations. Take the amount of labor and capital as given (that is, as exogenous variables). Choose one price as the numeraire (say we fix $p_{(urban)} = 1$). Delete the corresponding market clearing equation (in this case, deleting the linkage equation $y_{(urban)} = \sum_f t_{(urban,f)}$ will do the job). Then we are left with a model with 36 endogenous variables and 36 equations. The GAMS representation of this model is shown below.

```
$TITLE SAM
options limrow = 4;

SETS
i general index /labor, capital, rural, urban, food, clothing/
s(i) sectors /food, clothing/
f(i) factors /labor, capital/
h(i) households /rural, urban/;
ALIAS (i,ip);
ALIAS (i,iq);

PARAMETERS
b(s) technical coefficients
a(i,ip) share coefficients;
b('food') = 1.2;  b('clothing') = 1;
```

```
TABLE sam(i,ip)
           labor capital rural urban food clothing
labor                                 75      85
capital                               50      60
rural      90     30
urban      70     80
food                        60    65
clothing                    60    85             ;

a(i,ip)= sam(i,ip) / sum(iq, sam(iq,ip));
DISPLAY a;

POSITIVE VARIABLES
p(i) price
q(i) quantity
y(i) income
t(i,ip) payment
c(i,ip) commodity ;

VARIABLES
j  performance index;

EQUATIONS
eph(h)
eqs(s)
eys(s)
eyf(f)
eyh(h)
etfs(f,s)
ethf(h,f)
etsh(s,h)
eetsh(s,h)
ecfs(f,s)
eeys(s)
eeyf(f)
eeyh(h)
jd performance index definition;

* performance index equation
jd..    j =E= 0;

*sectors
eqs(s)..      q(s) =E= b(s)* prod(f, c(f,s)**a(f,s));
ecfs(f,s).. c(f,s) =E= a(f,s) * q(s) * p(s) / p(f);
eys(s)..      y(s) =E= p(s) * q(s);
etfs(f,s).. t(f,s) =E= p(f) * c(f,s);
```

```
*factors
eyf(f)..      y(f) =E= p(f) * q(f);
ethf(h,f)..  t(h,f)=E= a(h,f) * y(f);
*households
etsh(s,h)..       t(s,h) =E= a(s,h) * y(h);
eph(h)..             p(h)=E= prod(s, p(s)**a(s,h));
eetsh(s,h)..       t(s,h)=E= p(s) * c(s,h);
eyh(h)..            y(h) =E= p(h) * q(h);
*linkage
eeys(s)..             y(s) =E= sum(h,t(s,h));
eeyf(f)..             y(f) =E= sum(s,t(f,s));
eeyh('rural').. y('rural') =E= sum(f,t('rural',f));
*notice that we eliminate one linkage equations(Walras law)

*initial values to facilitate solver convergence
p.l(i) = 1;  q.l(i) = 1;  y.l(i) = 1;

*lower bound to avoid division by zero
p.lo(f) = 0.001;

*lower bounds to avoid undefined derivative in exp functions
p.lo(s) = 0.001;  c.lo(f,s) = 0.001;

*exogenous variables
q.fx('labor') = 2;  q.fx('capital') = 1;

*numeraire
p.fx('urban') = 1;

MODEL SAMDK /all/;
option iterlim = 10000;
SOLVE SAMDK MAXIMIZING J USING NLP;

PARAMETER REPORT;
REPORT(i, "price") = p.l(i);
REPORT(i, "quantity") = q.l(i);
REPORT(i, "income") = y.l(i);

DISPLAY REPORT;  DISPLAY t.l, c.l;
```

The GAMS representation is similar to the simple General Equilibrium model presented before. Here we make use of sets and subsets as indices, we use the ALIAS command to redefine an index so we can use it to index a matrix, we input the SAM as a table under the PARAMETER section, and we define indexed variables and equations. Notice

that, in order to have a more compact representation, we were able to use a general index "`i`" for variables, and later work with subsets of variables, but we did not do so for equations. GAMS does not admit the use of subsets as indices of equations.

As in the previous example, we should check that only nominal variables change (proportionally) when we change the numeraire.

4.2 A Johansen Style Model

CGE models tend to be large and nonlinear. As they grow in size, obtaining convergence (that is, a numerical solution) is likely to become more difficult. An alternative is to switch to a model representation pioneered by Leif Johansen (1960). Johansen style models are solved in a linearized form where all the variables are rates of growth. This method consists in transforming all the variables in the model into percentage changes with respect to a base case.

For example, given an expression in levels like

(14) $$X = a\ Y\ Z$$

if we first take logs, we obtain

(15) $$\log X = \log a + \log Y + \log Z$$

and totally differentiating

(16) $$d(\log X) = d(\log a) + d(\log Y) + d(\log Z)$$

that is (since a is a constant)

(17) $$\frac{dX}{X} = \frac{dY}{Y} + \frac{dZ}{Z}$$

or

(18) $$x = y + z$$

where x, y and z variables are percentage deviations.[21] In a similar fashion, we can transform

[21] An alternative derivation without using logs is as follows:

(19) $$X = a\ Y^b$$

into

(20) $$x = b\ y\ .$$

Thus for an expression like

(21) $$X = Y + Z$$

we totally differentiate

(22) $$dX = dY + dZ$$

then divide by the right hand side variable

(23) $$\frac{dX}{X} = \frac{dY}{X} + \frac{dZ}{X}$$

Then multiply and divide each term on the right hand side by the variable in its numerator and rearrange to obtain

(24) $$\frac{dX}{X} = \frac{dY}{Y}\frac{Y}{X} + \frac{dZ}{Z}\frac{Z}{X}$$

or

$$\frac{dX}{X} = \frac{Y}{X}\frac{dY}{Y} + \frac{Z}{X}\frac{dZ}{Z}$$

or

(25) $$x = s_y y + s_z z$$

$dX = YZda + aZdY + aYdZ$

$dX = aZdY + aYdZ$

$\frac{dX}{X} = \frac{aZdY}{X} + \frac{aYdZ}{X}$

$\frac{dX}{X} = \frac{dY}{Y} + \frac{dZ}{Z}$

where s_y and s_z are the shares $s_y = \frac{Y}{X} = \frac{Y}{Y+Z}$ and $s_z = \frac{Z}{X} = \frac{Z}{Y+Z}$.

In short, the transformation of a model in levels into one in percentage changes can, in many cases, be achieved by applying some simple rules. Given *X*, *Y* and *Z* as variables in levels, *a* and *b* as parameters and *x*, *y* and *z* as variables in percentage deviations, some useful rules are

(26) $X = a\ Y\ Z$ becomes $x = y + z$

(27) $X = a\ Y^b$ becomes $x = b\ y$

(28) $X = Y + Z$ becomes $x = s_y y + s_z z$

where s_y and s_z are the shares $s_y = \frac{Y}{Y+Z}$ and $s_z = \frac{Z}{Y+Z}$.

Applying these rules to the simple General Equilibrium model presented in Section 3 and interpreting each variable not as levels but as percentage changes with respect to a base case, we obtain the following GAMS representation

```
$TITLE JohansenGE

SCALARS
a labor share / 0.7 /

VARIABLES
qs good supply
qd good demand
ld labor demand
ls labor supply
kd capital demand
ks capital supply
p price
w wage
r profit
y income
j  performance index;

EQUATIONS
eqs good supply equation (production function)
```

```
eqd good demand equation
eld labor demand equation
els labor supply equation
ekd capital demand equation
eks capital supply equation
ey  income equation
eml labor market clearing
emk capital market clearing
jd performance index definition;

jd..       j =E= 0;

eqs..     qs =E= ld * a + kd *(1-a);
eld..     ld =E= qs + p - w;
els..     ls =E= 0;
eml..     ld =E= ls;
ekd..     kd =E= qs + p - r;
eks..     ks =E= 0;
emk..     kd =E= ks;
ey..       y =E= (0.7)*(w + ld)+ 0.3 *(r + kd);
eqd..     qd =E= y - p;

*numeraire
p.fx = 0;

MODEL JOHANSENGE /all/;
SOLVE JOHANSENGE MAXIMIZING J USING LP;
DISPLAY qs.l,qd.l,ld.l,ls.l,kd.l,ks.l,p.l,w.l,r.l,y.l;
```

Notice that we eliminated the *b* parameter from the scalars section, since we do not use it here. Also, notice that since percentage changes can be positive or negative, we no longer define the model variables as positive variables as we did in the version of the model where variables where in levels. Finally, notice that the values of the stock of labor and capital and the numeraire are equal to zero, since they are percentage changes. The 0.7 and 0.3 coefficients that appear in equation "ey" are the corresponding share parameters obtained when applying the third rule. Finally, we solve the model invoking a Linear Programming solver, since the problem is a linear one.

An interesting exercise is to compare the results of the nonlinear model in levels versus the linear model in percentage changes for a given change in an exogenous variable. For example, say we increase the stock of capital by 20 percent. This means that in the nonlinear model *k* goes from 1 to 1.2, while in the linear model it goes from zero to 0.2. The results are shown in Table 8.4.

	The Nonlinear Model			**The Linearized Model**
variable	solution k = 1	solution k = 1.2	percentage change	percentage change
q	1.949	2.059	5.6	6
l	2	2	0	0
k	1	1.2	20	20
w	0.682	0.721	5.7	6
r	0.585	0.515	-12	-14
y	1.949	2.059	5.6	6

Table 8.4 Comparison of Nonlinear and Linearized Models

The differences between the last two columns give us an idea of the approximation error of the linearized solution. We should expect this error to be larger the greater the change in the exogenous variables. Also, notice that if we simultaneously change the value of more than one exogenous variable for the linear version, the superposition principle will apply: the combined effect of changes in more than one exogenous variable will be equal to the sum of the individual effects

As we said above, solving nonlinear models may become problematic as they grow in size. The problem we just linearized using Johansen's technique is a very small one, and we used it to provide a simple illustration of the methodology. For an application to a larger model you are referred to Kendrick (1990), who provides a Johansen style GAMS representation of a version of the ORANI model developed by Dixon, Parmenter, Sutton and Vincent (1982) in Project Impact in Australia.

6. Experiments

For the input-output model in Section 1 you may perform experiments changing the levels of final demand, the values of some input-output coefficients or the nature of the capacity constraint restrictions.

For the production prices model in Section 2, an interesting experiment would be to pick one price as the numeraire (say $p_1 = 1$) and a technology such that the proportions between labor costs and total input costs is the same for each industry, that is, when the input-output coefficients are proportional for all industries. For instance, when the input-output matrix is

$$A = \begin{bmatrix} 0.05 & 0.025 & 0.1 \\ 0.1 & 0.05 & 0.2 \\ 0.2 & 0.1 & 0.4 \end{bmatrix}$$

and the labor coefficients vector is

$$L = \begin{bmatrix} 1/7 \\ 2/7 \\ 4/7 \end{bmatrix}$$

you will observe that prices will not change as *r* and *w* change in an inverse relationship.

For the small general equilibrium model in Section 3 the economy-wide effects of technological progress can be simulated by increasing the value of the *b* parameter. Also, you could change the supply of labor or the supply of capital and see how the wage and the profit levels are affected. If you do so, you will observe that quantities do not change, only the wage and the profit rate do. Quantities would change if you specified elastic labor and capital supply functions, instead of the fixed supplies assumed in the model. Also, we imposed the market clearing condition in all three markets. However, it may well be the case that that condition may not be appropriate for some markets because they are in "disequilibrium". That may happen, for example, because their prices are exogenously fixed. For such cases we should follow an appropriate modeling strategy such as the ones proposed, for example, by Malinvaud (1977).

Finally, for the SAM based CGE model in Section 4.1, you can perform interesting experiments by changing the amount of labor or capital or the technology parameters. Notice that you could also change the share parameters by changing some numbers in the

SAM. If you do so, remember to maintain the corresponding balance between rows and columns. Also, an interesting exercise would be to expand the model to incorporate foreign trade as in Kendrick (1990).

7. Further Readings

Dervis, de Melo and Robinson (1982) and Dixon, Powell, Parmenter and Wilcoxen (1992) provide extended textbook presentations of the different types of models introduced in this chapter. For historical and analytical presentations of input-output and production prices models see Pasinetti (1977) and for CGE models see Dixon and Parmenter (1996). Shoven and Walley (1992) deal extensively with neoclassical type CGE models, while Taylor (1990) presents neo-structuralist type CGE models. Roland-Holst, Reinert and Shiells (1994) provide an analysis of the North American Free Trade Area. Lofgren, Lee Harris and Robinson (2002) develop a standard CGE model in GAMS. For the use of a dynamic CGE model in a control context to study income distribution changes, see Paez (1999). For an approach to solving dynamic CGE models with stochastic control theory methods see Kim (2004).

Chapter 9

Cournot Duopoly in Mathematica

with

Daniel Gaynor

Students of economics are introduced first to the market structures of pure competition and of monopoly. However, most real world examples are in the domain of oligopolies that lie between these two extremes.

What distinguishes oligopolistic markets from either purely competitive markets or monopoly markets is that in an oligopoly market there is an interdependency of actions between firms. By interdependency, we mean that the choice of a given firm will affect and be affected by the choices of the other firms. This issue of interdependency does not exist in purely competitive markets or in monopolistic markets.

In a purely competitive industry, firms are assumed to be too small to influence the market price and therefore the action of one firm has no way of influencing (or being influenced by) the actions of another firm. Alternatively, a monopolist has tremendous influence over the market price, but as a monopolist, the firm has no other firms whose actions it can influence. In an oligopolistic industry, there are two or more firms competing in the market, each of which has the ability to influence the market price and therefore the choices of its competitors.

Problems involving interdependency of actions between multiple players are called games and game theory is the study of these multi-player decision problems[22]. We will use the Mathematica programming language to solve several alternative game theoretic models of oligopoly market structure. All of these models will be called *quantity* games since the strategic choice of the firms will be quantity. Alternative

[22] For a more comprehensive introduction to game theory see Gibbons (1992). Some of the examples used in the following are drawn from Gibbons' text.

models of price competition, like the Bertrand model in which each player chooses a price, will be discussed briefly, but not modeled. We will focus here on two-firm oligopoly models, called *duopolies*, but the models can easily be extended to incorporate a larger number of firms (which the student is encouraged to do in the Experiments section).

While all of the problems discussed here can be solved with pen and paper, the use of Mathematica opens the door to the solution of substantially more complex models.

The three topics covered here are game theory, oligopoly market structure and the Mathematica programming language. Expertise in any of these individual areas should enhance the learning of the other two areas. However, the material covered in all three of these areas is kept at an introductory level and no previous knowledge is required. We begin with a short introduction to game theory as a means of introducing the tools and terminology that are required for our oligopoly models. Then we will examine two popular models of oligopoly market behavior using Mathematica to derive their results and discuss the intuition of the solutions. In this chapter we will discuss the Cournot model and in the next chapter the Stackelberg model.

1. Game Theory

By identifying our oligopoly markets as games, we have already gone further than you might think towards modeling and solving these models. After all, game theory tells us how we can represent a game as well as how one should approach solving a game. In addition to introducing some basic concepts of game theory, this section will discuss a very simple but popular game called the Prisoners Dilemma. The Prisoners Dilemma game is an extremely valuable tool because there is a direct parallel between this simple game and the oligopoly games we plan to solve.

There are many types of games, but it is useful to distinguish between a few. To begin, games can be either *simultaneous move* or *sequential move*. In a simultaneous move game, all of the players choose their actions simultaneously without observing each other's actions. In a sequential move game there is an order to the play. More precisely, a sequential move game is dynamic in that one player chooses an action, and then a second player chooses an action only after observing the first player's action.

A second classification of games is *complete information* versus *incomplete information* games. All of the games presented in this chapter are complete information games or games in which no firm has private information about itself that other firms do not have

access to. Finally, it will be helpful to distinguish between games with *discrete* strategy choices versus games with *continuous* strategy choices.

The first step in approaching a game is representing the game. There are two ways of representing a game: the normal form (usually a two-way table, appropriate for simultaneous move games) and the extensive form (usually a game tree, more appropriate for sequential move games). We adopt here the normal-form representation of a simultaneous play game. The three elements that constitute normal-form representation of a game are: 1) the players participating in the game, 2) the strategies (or actions) that are available to each of the players, and 3) the payoffs that each player would receive for each possible combination of strategies chosen by the players. For two-player games in which the strategy choices are discrete, normal-form games can be represented in table format as the Prisoners Dilemma game in Table 9.1[23].

		Player II	
		Mum	**Fink**
Player I	**Mum**	-1 , -1	-9 , 0
	Fink	0 , -9	-6 , -6

Table 9.1 The Prisoners' Dilemma Game

Notice that Figure 9.1 completely represents the Prisoners Dilemma game according to our definition of a normal-form representation. First there are two players involved in the game - Player I and Player II. Second, Player I can choose between the strategies Mum and Fink and similarly Player II can choose between the strategies Mum or Fink. Finally, the payoffs from each of the possible combinations of Player I/Player II strategies are represented by the table's payoff matrix. For example if Player I plays Mum and Player II plays Fink, then Player I receives a payoff of negative nine (or nine years in jail) and Player II receives a more favorable payoff of zero.

The story of the Prisoners Dilemma game is as follows. Two suspects of a crime are detained by the authorities and interrogated separately. Each player can either offer no information (Mum) or can blame the crime on the other player (Fink). Furthermore, prisoners must choose their strategies without observing each other's choice. If both players choose Mum each only spends one year in jail (1 unit of negative utility). If each

[23] See Gibbons (1992), p. 3.

blames the other (Fink, Fink) then each spends six years in jail. If one player Finks and the other player chooses Mum, then the player who finks goes free and the player who chose Mum spends nine years in jail.

In order to solve the Prisoners Dilemma, we adopt the notion of *Nash Equilibrium Strategies*. The strategies of a game's players constitute a Pure Strategy Nash equilibrium (called Nash equilibrium here-after)[24] if each player's chosen strategy is the best response to the strategies played by all of the other players. In other words, a Nash equilibrium occurs when every player chooses his strategy optimally given his opponents' chosen strategies.

Applying the concept of Nash equilibrium strategies to the Prisoners Dilemma we can find the Nash equilibrium. Begin by considering how Player II would best respond to Player I playing the strategy Mum. In this event, Player II could also play Mum and spend a year in jail or Player II could play Fink and go free. So Player II's best response to Player I choosing Mum is Fink. Therefore, the strategy (Mum, Mum) does not constitute a Nash equilibrium.

Does Player I choosing Mum and Player II choosing Fink (Mum, Fink) constitute a Nash equilibrium? The answer is no, because while Fink is Player II's best response to Player I playing Mum, in order to be a Nash equilibrium it must also be the case that Mum is Player I's best response to Player II playing Fink. However, we see that if Player II Finks, Player I's best response is to Fink as well. Continuing with this logic you will find that the only Pure Strategy Nash Equilibrium for this game is for both players to fink on each other (Fink, Fink).

Through this example, we have barely scratched the surface of game theory. However, we have addressed a few of the basics that will allow us to better understand oligopoly market structure. First, we know the three elements that constitute a simultaneous game are: the players of the game, the strategies (or choices) available to each of these players, and each player's payoff for every possible combination of players' strategies. A forth element which we will see is important in sequential move games is determining the order of play. These are necessary for characterizing and solving any game. Finally, our simple example illustrated the solution concept that we will employ in our oligopoly problems - the pure strategy Nash equilibrium. The intuition of Nash

[24] In game theory there is a distinction between Pure Strategy Nash equilibria and Mixed Strategy equilibria. This is a distinction that is beyond the scope of this chapter except to note that by Nash equilibria we mean Pure Strategy Nash equilibria.

equilibrium is that each player is choosing his best response or reaction to all of the other players' choices.

2. Static Models of Oligopoly Markets

Because models of oligopoly markets depend on how firms interact, characteristics of the environment in which they interact, and potentially many other factors, there is no single model of oligopoly market structure. The correct model will depend on the characteristics of the industry being modeled. We focus here on two "one-shot" quantity games that are the foundations for many more sophisticated models of oligopoly markets. These models are one-shot in the sense that the game is only played a single time, not repeatedly played every period. They are referred to as quantity games because the strategic choices of the firms' are their respective outputs (or quantities). As we will see, quantity games have an interesting characteristic that we will exploit: If the quantity choices are assumed to be continuous, then the payoffs will also be continuous. Natural alternatives to quantity games are pricing games. Models of price competition have a winner take all aspect where the firm who has the lowest price captures the entire market (and the market is split in the event of a tie). Therefore, while the strategic variable price is continuous, the payoffs (profits) are discontinuous. We do not consider pricing models except to note that the solutions to such games will vary significantly from the quantity games considered here.

3. Cournot Competition

The first model of oligopoly market structure that we will study is the model of Cournot quantity competition named for the French mathematician, Augustin Cournot. Cournot first presented the model in his book, *Researches into the Mathematical Principles of the Theory of Wealth*, published in 1838 one hundred and twelve years before John Nash formalized the concept of Nash equilibrium strategies.

To make the problem more tractable, but without loss of generality, we will assume that the industry is a duopoly. Our story of a Cournot duopoly market is as follows: There is a market consisting of two firms producing a homogenous good each at a constant (not necessarily the same) marginal cost. We assume that each firm knows its own cost as well as its competitor's cost and that they also know the market's demand function

$$Q = f(p)$$

where Q is quantity and p is price. Also we assume that they can derive the inverse demand functions

$$p = g(Q)$$

The problem faced by these two firms is that each must choose the quantity that it will supply to the market without observing its competitor's output choice, and let the market determine the price. This requires each firm to anticipate, when choosing its own quantity, how the other firm will behave.

The game described above is a one-shot simultaneous move game and as such can be represented as a normal form game. To do so we need to identify the three necessary ingredients. First, we have the players; the two firms who we will denote Firm 1 and Firm 2. Next, we must identify the firms' strategic choice. As noted, the firms' strategic choice is quantity, which is assumed to be a continuous non-negative variable. Finally, the payoffs to each of the players in the game is simply the profit that this firm earns given its choice of quantity and the quantity chosen by the other firm.

To solve this game we will apply the same solution concept we used to solve the Prisoners' Dilemma game. A pure strategy Nash equilibrium for this Cournot game is a set of quantities

$$\left(Q_1^*, Q_2^*\right)$$

in which each of the firms chooses its profit maximizing output given its forecasted output choice of the other firm, and each firm's forecast of the other firm's output is correct. Recall that in the discrete strategy prisoners' dilemma game, finding the Nash equilibrium required us to consider each possible combination of strategies. However, in the Cournot game each firm has a continuum of possible strategy choices and therefore there are an infinite number of possible combinations of players' strategies. Fortunately, in the continuous Cournot model, we can generalize the strategic behavior of the firms by deriving what we will call a reaction (or alternatively a best response) function. As we will see, calculus permits us to do this because our firms' payoffs (their profits) are continuous functions of their own quantity choice as well as the other firm's quantity. To see this more clearly, we will begin with a Mathematica program (`react.nb`) that is available on the book web site. The program illustrates strategic behavior and the solution to the Cournot model graphically. The focus of this experiment is to familiarize the reader with the concept of a reaction function and to understand its connection to determining Nash equilibria. The instructions for running Mathematica are in Appendix

B. Turn to that appendix and run the `react.nb` file if you prefer to follow along the Mathematica code while you are reading the rest of this chapter.

The first step in building the graphical model of the Cournot game is to model the industry characteristics. Because the market is a duopoly, total market quantity, Q, is the sum of firm 1's output choice, Q_1, and firm 2's output choice, Q_2, i.e.

(1) $$Q = Q_1 + Q_2$$

This simple assignment is made in Mathematica with the following input statement

```
IN[]:=      Q = Q1 + Q2;
```

The symbols `IN[]:=` are the Mathematica prompt for input and the expression `Q = Q1 + Q2;` is the user's input. It is important to note that the equal sign in the input is used in Mathematica for assigning names to expressions (or assigning values to variables). It does not define a formula or equation. To create an equation, you must use two consecutive equal signs (==). Also note the use of the semicolon at the end of the above command. The semicolon is used to suppress the display of output created by Mathematica for each input statement.

The other general market characteristic that we must specify at this time is the functional form for the inverse market demand faced by the duopolists. In all of our models we assume that the inverse demand curve is linear, i.e.

(2) $$\text{Price} = a - bQ$$

or in Mathematica

```
IN[]:=      Price = a - b*Q
```

Because the Cournot game is a simultaneous choice game, it does not matter whether we consider firm 1's or firm 2's optimization problem first. Therefore we will consider firm 2's problem first.

Firm 2's profits are equal to the difference between the revenue from selling the quantity Q_2 and the cost of selling this quantity. Revenue is firm 2's own quantity (Q_2) multiplied by the market price, i.e.

(3) $$\text{Profit}_2 = Q_2\left(\text{Price} - c_2\right)$$

where c_2 is firm 2's constant unit cost. This can be written in Mathematica as

```
IN[]:=      eqPr2 = Profit2 == Q2(P[Q1,Q2] - c2) ;
```

Notice that the price, P, is a function of the quantity decisions of the two firms, i.e. Q_1 and Q_2. The above statement creates a formula named eqPr2 that defines Profit2 (firm 2's profits) to be equal to Q_2 multiplied by the difference between market price and the firm's marginal cost. We have used the string

```
P[Q1,Q2]
```

here to indicate that the price is a function of both Q1 and Q2; however, Mathematica does not recognize the functional dependency. Rather it just treats P[Q1,Q2] as string of characters.

This is the general representation of firm 2's profits. However, because we have specified that our industry is subject to a linear demand curve, we want to replace the general form of the price function with the linear demand specified in the earlier Mathematica statement

Price = a - bQ

While the practice of defining a generalized profit function and then substituting in a specific functional form may seem cumbersome, it is a good programming practice because it allows us to change the functional form of our market demand by editing a single Mathematica statement.

Mathematically we obtain an expression for the profit of the second firm by substitution of Eq. (2) into Eq. (3) to obtain

(4) $$\text{Profit}_2 = Q_2\left[a - bQ - c_2\right]$$

and then substituting Eq. (1) into Eq. (4) to obtain

(5) $$\text{Profit}_2 = Q_2\left[a - b\left(Q_1 + Q_2\right) - c_2\right]$$

These steps are accomplished in Mathematica in the following way. First the substitution of the specific linear demand function for the general form is done with the Mathematica statement

```
IN[]:=      eqPr2 = Expand[% /. P[Q1,Q2] -> Price]
```

In Mathematica, % refers to the last result generated (and %% refers to the 2nd to last result generated etc.) and /. is the replacement identifier. So the above statement takes the original profit equation (named eqPr2) and replaces the general form of the demand equation `P[Q1,Q2]` with our explicit linear inverse demand expression "`Price`" which is specified above (`Price = a - b*Q`). Since no semicolon is used at the end of the statement above, the output statement gives the result of this substitution.

```
OUT[]:=     Profit2 == Q2 (a - c2 - b (Q1 + Q2))
```

This output is the equation for firm 2's profits or payoff as a function of the firms' quantities.

To find firm 2's profit maximizing behavior, we take the derivative of its profit function, Eq. (5), with respect to its choice variable, Q_2, and set the expression equal to zero, i.e.

$$\frac{\partial \text{Profit}_2}{\partial Q_2} = a - c_2 - b\left(Q_1 + Q_2\right) + Q_2\left(-b\right) = 0 \tag{6}$$

In our Mathematica program this is accomplished below by defining a new equation, which we will name focPr2 (first order condition for profit of firm 2). It is the derivative, D[ ], of firm 2's profit (payoff) function with respect to its choice of its own quantity Q_2.

```
IN[]:=      focPr2 = D[eqPr2, Q2]
```

This will produce the following output that is the derivative of equation `eqPr2` (after the substitution) with respect to `Q2`.

```
OUT[]:=     0 == a - c2 - b Q2 - b (Q1 + Q2)
```

This first order condition implicitly describes firm 2's optimal behavior. However, we want to find an explicit solution for describing firm 2's optimal behavior.

This is done by solving the first order condition in Eq. (6) for Q_2, i.e.

(7) $$2bQ_2 = a - c_2 - bQ_1$$

or

(8) $$Q_2 = \frac{a - c_2 - bQ_1}{2b}$$

This is accomplished in Mathematica by using the `Solve` statement to solve the first order condition for `Q2` and then naming the output `temp2`. It is helpful to note that the output from a solve statement is a list of solutions. Consequently, `temp2` is the name of the list of solutions. In the case below, the list `temp2` has only one solution and therefore a single element.

```
IN[]:=    temp2 = Solve[focPr2, Q2]
OUT[]:=   {{Q2 -> (a - c2 - b Q1)/(2 b)}}
```

It is clear from the first order condition above that firm 2's optimal choice of Q_2 will depend on firm 1's optimal choice Q_1. This is the key to game theoretic problems; each party must consider what the other parties will do. Because firm 2's choice of Q_2 is a function of firm 1's choice of Q_1, we call the expression above firm 2's "Best Response" or "Reaction Function". As its name implies, this function dictates how firm 2 chooses Q_2 as a best response (or in reaction) to firm 1's choice of Q_1. In the simply linear case, we can solve for firm 2's best response quantity (`R2[Q1]`) explicitly.

In the next line of code, we create an expression called `React2` (firm 2's reaction function) that represents firm 2's optimal response (`R2[Q1]`) to firm 1's quantity choice `Q1`.

```
IN[]:=    React2 = R2[Q1] == Q2 /. temp2[[1]]
OUT[]:=   R2[Q1] == (a - c2- b Q1)/(2 b)
```

The right hand side of this expression is simply the solution for Q_2 that we found above. In other words, `R2[Q1]` is equal to `Q2` where `Q2` is replaced (/.) by the first solution (

[[1]]) from the output list "temp2". R2[Q1] is just another name for Q2 that reflects the fact that this quantity is chosen in response to Q1.

We conclude our examination of firm 2's behavior in the Cournot Duopoly model by graphing the reaction function for firm 2 that has been derived from the model above.

```
IN[]:=
reactPlot =
Plot[{Q2 /. Solve[focPr2 /. {a -> 1, b -> 1, c2 -> .5},Q2][[1]]},
     {Q1, 0, .55},
     PlotRange -> {0, .55},
     PlotStyle->{RGBColor[1,0,0],Thickness[0.010]},
     AxesLabel->{"Q1","Q2"},
     PlotLabel->"Reaction Curve" ]
```

This rather messy looking Mathematica command creates a plot that we name reactPlot using the Plot command. The syntax for the Plot command is

```
Plot[f, {x, xmin, xmax}, option -> value]
```

where f is the expression to be plotted, the list {x, xmin, xmax} specifies the minimum and maximum values that the variable in the expression takes, and option -> value statements are used to set any display attributes of the graph.

Starting from the second line of code above following the IN[]:= statement, the Plot[] command is used to create the plot. The function, f that we want to plot is an expression that represents the values that Q2 takes - expressed in terms of the variable Q1 and the model's parameters, i.e.

```
Q2 /. Solve[focPr2 /. {a -> 1, b -> 1, c2 -> .5},Q2][[1]]
```

Our identification of firm 2's reaction equation has shown that in the linear demand case we can find such an expression by solving the first order condition, focPr2, for Q2. Therefore, this line of code tells Mathematica that we want to plot the values of the variable Q2, where an expression for Q2 is found from solving the first order condition of firm 2's profit function for the variable Q2, i.e.

```
Solve[focPr2, Q2]
```

Within this solve statement there is a replacement command (/.) followed by a list ({,}) of replacements. These replacements specify the specific numerical values the models parameters are assumed to take in this example.

The next line of this plot command {Q1,0,.55} specifies the range of values for the variable Q1 in the expression for Q2. The remaining lines of code specify display options for the Mathematica plot. Options are used to control the plot color and thickness, axes labels, and plot labels. The resulting plot shows what quantity, Q_2, is firm 2's best response to any given quantity of by firm 1 (Q_1).

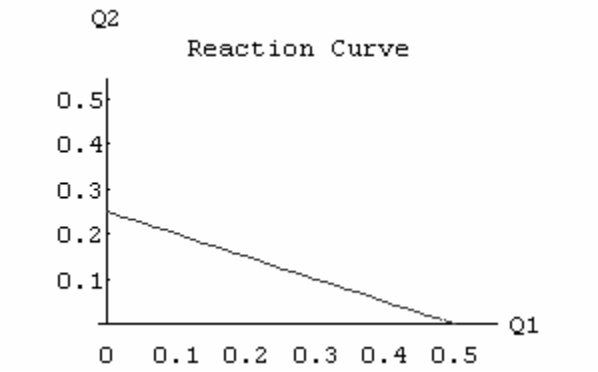


Figure 9.1 Reaction Curve of Q_2 to Q_1

Thus if firm 1 chooses $Q_1 = 0.3$ then firm two's optimal reaction is to choose $Q_2 = 0.1$.

Next we turn to the optimization problem for firm 1. It solves a problem that is identical to firm 2's except for the fact that firm 1 solves for his own quantity Q_1 and has a marginal cost of c_1. The solution to firm 1's reaction function is

$$\text{OUT[]:=} \quad \text{R1[Q2]} == \frac{\text{a - c1- b Q2}}{\text{2 b}} .$$

We can plot this relationship for firm 1 assuming the same parameter values and over the same interval of values as we did with firm 2.

```
IN[]:=
      reactPlot =
      Plot[{Q2 /. Solve[focPr1 /.{a -> 1, b -> 1, c1 -> .5}, Q2][[1]]},
           {Q1, 0, .55},
           PlotRange -> {0, .55},
           PlotStyle->{RGBColor[0,0,1],Thickness[0.001]},
           AxesLabel->{"Q1","Q2"},
           PlotLabel->"Reaction Curve"]
```

In the graphical illustration of the Cournot solution shown below we are assuming that the firms have identical cost structures ($c_1 = c_2 = 0.5$). This plot shows what quantity, Q_1, is firm 1's best response to any given quantity choice by firm 2 (Q_2).

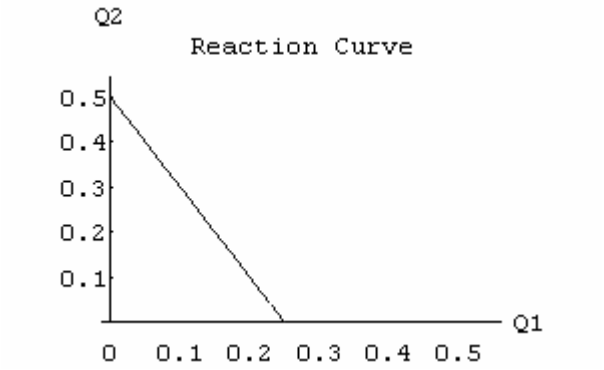


Figure 9.2 Reaction Curve of Q_1 to Q_2

At this point we have plots showing how each firm should best respond to it competitors various choices of output. According to our definition of Nash equilibria, a Nash equilibrium of this game is a set of strategies in which each firm is choosing an output that is a best response to the other firm's output choice. Or more succinctly, at a Nash equilibrium, both players will be on their reaction functions. Before looking at this graphically, however, it is useful to see how the costs affect the solutions to our simple Cournot model. The simplest and most intuitive way to investigate this consideration is with another Mathematica plot.

```
IN[]:=
reactPlot =
Plot[{Q2 /. Solve[focPr1 /.{a -> 1, b -> 1, c1 -> .5}, Q2][[1]],
      Q2 /. Solve[focPr1 /.{a -> 1, b -> 1, c1 -> .6}, Q2][[1]]},
 {Q1, 0, .55},
 PlotRange -> {0, .55},
 PlotStyle->{
      {RGBColor[0,0,1],Thickness[0.001],},
      {RGBColor[0,0,1],Thickness[0.001],Dashing[{.03,.02}]}
      },
 AxesLabel->{"Q1","Q2"},
 PlotLabel->"Reaction Curves"]
```

Notice the syntax of the Mathematica statement above. Because we are plotting two reaction functions the first element in the `Plot[]` command becomes a list of expressions `{f1, f2}` where the first element in the list is firm 1's reaction function when its marginal cost is 0.5 and the second is firm 1's reaction plot when its own marginal cost is 0.6. In the above,

```
RGBColor[0,0,1], Thickness[0.001]
```

indicates that the first plot will be a solid blue line[25] which is 0.001 thick and

```
RGBColor[0,0,1], Dashing[{.03,.02}]
```

indicates that the second line will be blue and have dashes of length .03 and spacing of .02. With the above statement we create a graphic that plots firm 1's reaction curve with the original parameter specifications and a marginal cost of 0.5 and then contains a second dashed plot that shows firm 1's reaction curve with a slightly higher marginal cost of 0.6. The color will show only in some printings of this book but will show in the online plot.

[25] Syntax: RGBColor[*red, green, blue*] where color intensities range from zero to one.

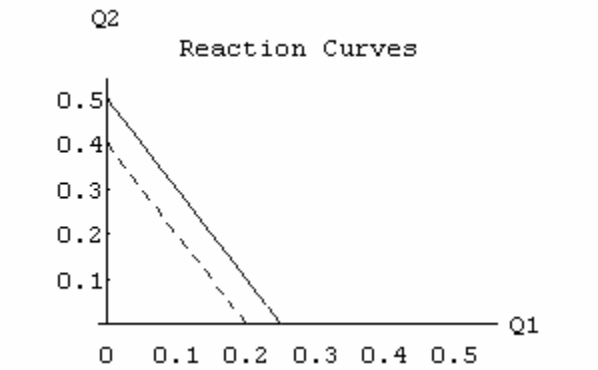


Figure 9.3 Sensitivity Analysis for the Reaction Curve of Q_1 to Q_2

From this graphical sensitivity analysis of firm 1's reaction functions we can see that higher unit costs, in the dashed line, will shift a firm's reaction function downward. That is, for a given output choice by firm 2, firm 1's best response quantity will be decreased for higher values of its own marginal cost. It is easy to see that firm 2's marginal cost, c_2, will have no effect on firm 1's reaction function since the Mathematica variable `c2` does not appear in firm 1's reaction function. Similarly, firm 1's marginal cost has no effect on firm 2's reaction function.

Our plots of the firms' reaction functions illustrate how each firm will choose its optimal quantity in response to the quantity choice of the other firm. If we knew firm 1's choice of Q_1 we could solve firm two's reaction function for its optimal choice of Q_2. Similarly, if we knew firm 2's choice Q_2 we could solve firm one's reaction function for its optimal choice of Q_1. The difficulty with the Cournot game is that the players simultaneously choose their respective quantities. Therefore, the solution to the simultaneous move Cournot game is found by solving both firms' reaction functions simultaneously for the quantities Q_1 and Q_2. Intuitively, the Nash equilibrium solution to the Cournot game has each firm choosing its best response quantity in reaction to the hypothesized quantity of the other firm[26]. To show the graphical solution to the Cournot model we plot both firms' reaction functions in a graph.

[26] See Gibbons (1992), p. 62.

```
IN[]:=
reactPlot =
Plot[{Q2 /. Solve[focPr1 /.{a -> 1, b -> 1, c1 -> .5}, Q2][[1]],
      Q2 /. Solve[focPr2 /.{a -> 1, b -> 1, c2 -> .5}, Q2][[1]]},
     {Q1, 0, .55},
     PlotRange -> {0, .55},
     PlotStyle->  {
                  {RGBColor[0,0,1],Thickness[0.001]},
                  {RGBColor[1,0,0],Thickness[0.010]}
                  },
     AxesLabel->{"Q1","Q2"},
     PlotLabel->"Reaction Curves"]
```

The Mathematica statement given above produces a graph showing the equilibrium strategies.

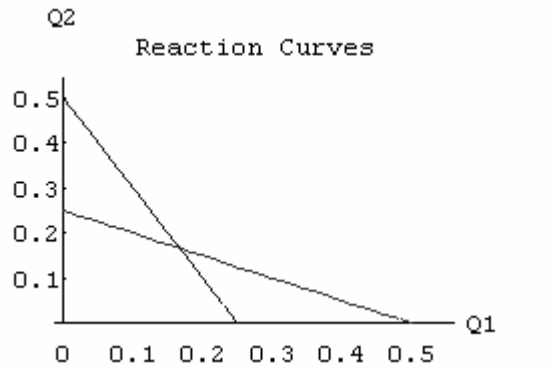


Figure 9.4 The Two Reaction Curves

At the point where the two firms' reaction functions intersect, each firm is choosing their respective profit maximizing output given their belief about the other firm's output choice and each of the firm's beliefs about the other is correct. This is the definition of a Pure Strategy Nash equilibrium.

This completes our use of the `react.nb` Mathematica file. While the graphical model above illustrates the behavior of the Cournot game's players in an intuitive way and clearly demonstrates how the Nash equilibrium is determined, we desire more from our model than intuition. The next Mathematica program (`cournot.nb`) is a model of the same Cournot duopoly. But in addition to a graphical solution, we will derive the

analytic solution to this problem. Solving for the Cournot-Nash quantities will permit us to determine the market supply and consequently the market price. In addition, with the Nash equilibrium quantities and the corresponding market price we can derive the firms' profit levels. All of this information is useful if we want to make comparisons between models of different market structures.

In the file `cournot.nb` we derive each firm's reaction function just as in the previous model. However, after identifying the firms' reaction functions, rather than using a plot to find the Cournot-Nash solution, we will solve for the optimal quantities directly. Recall from our graphical example that the Nash equilibrium strategy set was defined by the intersection of the two firms' reaction functions. Also recall that our definition of a Nash equilibrium requires that all players simultaneously give their best response to each other's choices. Clearly, the Nash equilibrium strategies can be found by simultaneously solving the set of reaction functions for the models strategic output choices.

To solve for the optimal quantities we add the following Mathematica statement,

```
IN[]:=
cournotQ =
Simplify[Solve[{React1 /. R1[Q2] -> Q1,
                React2 /. R2[Q1] -> Q2},{Q1, Q2}]]
```

The above command renames each of the firms' reaction quantities `Ri[Qj]` with the firm's actual chosen quantity `Qi` and then solves the two equations simultaneously for the choice variables `Q1` and `Q2`. The resulting output from the command is the Nash equilibrium strategy.

$$\text{OUT[]:=}\quad \{\{Q1 \to \frac{a - 2\,c1 + c2}{3\,b},\ Q2 \to \frac{a + c1 - 2\,c2}{3\,b}\}\}$$

In order to save these results we create two new Mathematica variables, `Q1c` and `Q2c` for each of the respective firms Cournot quantities.

```
IN[]:=    Q1c = Q1 /.%[[1]] ;
IN[]:=    Q2c = Q2 /.%%[[1]] ;
```

The interpretation of these statements is; `Q1c` is defined as the variable `Q1` where `Q1` is replaced with the values from the first solution of the previous Mathematica output and similarly for `Q2c`.

In a similar manner we derive and store the Cournot market output, the Cournot market price, firm 1's profits, and firm 2's profits respectively.

```
IN[]:=    Qcour = Q /.{Q1 -> Q1c, Q2 -> Q2c}
```

$$\text{OUT[]:=}\quad \frac{a + c1 - 2\,c2}{3\,b} + \frac{a - 2\,c1 + c2}{3\,b}$$

In the above statement, the Cournot market output, `Qcour`, is defined to be the market output, `Q`, which was defined to be `Q1 + Q2` where we replace the `Q1` with firm 1's Cournot quantity `Q1c` and replace `Q2` with firm 2's Cournot quantity `Q2c`.

The Cournot market price, `Pcour`, is found by substituting the firms' Cournot outputs, `Q1c` and `Q2c`, in place of `Q1` and `Q2` into the inverse demand function `Price` which was defined at the start of the program, i.e.

```
IN[]:=    Pcour = Simplify[Price /.{Q1 -> Q1c, Q2 -> Q2c}]
```

$$\text{OUT[]:=}\quad \frac{a + c1 + c2}{3}$$

The Mathematica `Simplify` command is used in the above input statement to provide a simplified output expression.

With the market price determined, calculating firms' profits is straightforward. Firm 1's Cournot profit, designated pie1c, is the firm's Cournot output, `Q1c`, multiplied by the difference between the price and firm 1's unit cost, i.e. `(Pcour - c1)`.

```
IN[]:=    pie1c = Simplify[Q1c*(Pcour - c1)]
```

$$OUT[]:= \frac{(a - 2\ c1 + c2)^2}{9\ b}$$

A similar expression calculates firm 2's Cournot level of profits, `pie2c`.

```
IN[]:=    pie2c = Simplify[Q2c*(Pcour - c2)]
```

$$OUT[]:= \frac{(a + c1 - 2\ c2)^2}{9\ b}$$

4. Experiments

In this chapter we develop a series of experiments that cover many of the aspects of the models presented. However, one set of experiments to consider is the use of alternative cost functions and another is to consider modeling alternative market structures.

5. Further Reading

For an introduction to Mathematica see Wolfram (2003). For a more comprehensive introduction to game theory see Gibbons (1992). Some of the examples used in this chapter are drawn from that book so the reader will find continuity between this chapter and Gibbons book. For an introduction to the use of Mathematica in game theory see Dickhaut and Kaplan (1993). For a study on the use of Mathematica to simulate the effects of mergers among noncooperative oligopolists see Froeb and Werden (1996).

We turn next to a different approach to solving the oligopoly problem, namely the Stackelberg Leadership model.

Chapter 10

Stackelberg Duopoly in Mathematica

with

Daniel Gaynor

Stackelberg quantity games are similar to Cournot games in that both are quantity competitions[27]. In the Stackelberg game, however, firms do not choose quantities simultaneously. Rather, this is a two-stage model in which a dominant firm (or Stackelberg leader) moves first by choosing its level of output in the first stage. After observing the leader's move the other firm chooses its best response output in the second stage. As we will see, the sequential play will require a different methodology and produce different results than those of the simultaneous move Cournot game.

1. The Stackelberg Leadership Model

As is typical with sequential games, we will solve the Stackelberg game backwards. Thus we begin to solve the problem by characterizing how the Stackelberg follower (firm 2) will respond to the Stackelberg leader's choice of quantity. An intuition for this backward approach is that in order for the Stackelberg leader to make an optimal decision about his output choice, he must first consider how the Stackelberg follower will respond to his choice in the second period. Because firm 1, our Stackelberg leader, has the same information as firm 2, the follower, firm 1 can solve firm 2's optimization problem just as well as firm 2 can. Therefore, the Stackelberg leader will solve for the Stackelberg follower's reaction function and then anticipate firm 2's second period response when making its own output choice in the first period.

[27] See Varian (1993b), pp. 448-454.

In this model we use the same specification for market demand that was used in the Cournot model. See the Mathematica file `stack.nb` that, as is discussed in Appendix B, is accessed in the same way as the previous Mathematica files. The model of the Stackelberg follower's 2nd period behavior is identical to firm 2's behavior in the Cournot model with only a subtle difference. Recall from the Cournot game that firm 2's reaction function was interpreted as firm 2's best response to firm 1's hypothesized output Q_1. In the Stackelberg game, firm 2 knows Q_1 for certain since it observed Q_1 at the end of stage 1. Therefore, firm 2 will respond to firm 1's observed output Q_1 by producing:

```
IN[]:=  Q2s = Simplify[R2[Q1] /. Solve[React2, R2[Q1]] [[1]] ]
```

$$OUT[]:= \quad \frac{a - c2 - b\ Q1}{2\ b} \ .$$

It is important to recognize that in the statement above `R2[Q1]` is a reaction function for firm 2, named `React2`, which is the same reaction function for firm 2 that was found in the Cournot game. Since firm 2, the Stackelberg follower, has already observe firm 1's output choice, `Q1`, firm 2 will best respond by choosing the output `Q2s`.

The Mathematica statement takes firm 2's reaction equation in the form

```
lhs == rhs
```

(left hand side = right hand side) that is named `React2`, and transforms it into an expression, named `Q2s`, which can later be substituted into firm 1's optimization problem. This is accomplished by defining an expression for the output of firm 2 in the Stackelberg game, `Q2s,` to be equal to the variable `R2[Q1]` where `R2[Q1]` is replaced (**/.**) with the expression representing the solution for the variable `R2[Q1]` from firm 2's reaction equation, `React2`. The `Solve` command finds the expression for the variable `R2[Q1]` and the `Simplify` command is used again to simplify the output expression. The `[[1]]` term in the input statement tells Mathematica to use the first solution found by the `Solve` command. Although there is a unique solution to the above Solve statement, this term is still required.

After solving firm 2's optimization problem, we step back to the first stage and solve the Stackelberg leader's (firm 1's) optimization problem. As we did in the Cournot Model, we begin by specifying the general form of firm 1's profit function and then replace the general demand function with our specific linear demand function.

```
IN[]:=      eqPr1 = pie1s == Q1*(P[Q1,Q2]-c1)
OUT[]:=     pie1s == Q1 (-c1 + P[Q1, Q2])

IN[]:=      eqPr1 = %/. P[Q1,Q2] -> Price
OUT[]:=     pie1s == Q1 (a - c1 - b (Q1 + Q2))
```

However, at this point the model makes a departure from the Cournot model. Because firm 1 can solve for firm 2's best response quantity as well as firm 2 can, firm 1 will anticipate firm 2's reaction to any choice of Q_1. Therefore, firm 1 can substitute firm 2's reaction output (which expresses firm 2's optimal choice of Q_2 as a function of firm 1's quantity Q_1) in place of Q_2 leaving a profit function for firm 1 which is a function of only its own quantity. This is accomplished with the following command

```
IN[]:=      eqPr1 = Simplify[%/. Q2 -> Q2s]

OUT[]:=     pie1s == Q1(a - 2 c1  + c2 - b Q1)
                     ---------------------
                              2
```

The above input statement redefines the equation for the profits of firm 1 (`eqPr1`) to be the equation from the previous output statement with the variable `Q2` replaced with the expression for `Q2s` which was calculated earlier. At this point the equation representing the Stackelberg leader's profits is only a function of its own quantity choice, Q_1.

Observing the previous output statement, it is easy to see that the Stackelberg leader's optimal choice of quantity can then be found by differentiating the profit function above with respect to that firms' choice of quantity Q_1. The Mathematica statement below uses the derivative command, `D[],` to differentiate firm 1's profit equation, `eqPr1`, with respect to the variable `Q1`.

```
IN[]:=      focPr1 = Simplify[D[eqPr1, Q1]]

OUT[]:=     0 == a/2 - c1 + c2/2 - b Q1.
```

The Stackelberg leader's output choice, Q_1, can then be found by solving the above first order condition for the variable Q_1. In the statement below, the `Solve` command finds this expression for the variable `Q1`, simplifies the expression, and then names the expression `Q1s` (the optimal quantity of firm 1 playing the Stackelberg game). The solution to the Stackelberg game has the Stackelberg leader (firm 1) choosing its optimal quantity:

```
IN[]:=    Q1s = Simplify[Q1 /.Solve[focPr1, Q1][[1]]]
```

$$OUT[]:= \frac{a - 2\ c1 + c2}{2\ b}$$

Firm 2 (the follower) then takes firm 1's quantity choice as given and reacts by choosing:

```
IN[]:=    Q2s = Simplify[Q2s /. Q1 ->Q1s]
```

$$OUT[]:= \frac{a + 2\ c1 - 3\ c2}{4\ b}$$

The above statement simply takes the expression for `Q2s` and replaces the variable `Q1` with our expression for `Q1s` in terms of the models parameters.

The resulting market quantity and market price are then calculated by substituting the firms' optimal quantities into the quantity and price equations.

```
IN[]:=    Qstack = Q /. {Q1 -> Q1s, Q2 -> Q2s}
```

$$OUT[]:= \frac{3\ a - 2\ c1 - c2}{4\ b}$$

```
IN[]:=    Pstack = Simplify[Price /. Q -> %]
```

$$OUT[]:= \frac{a + 2\ c1 + c2}{4}$$

The syntax for the expressions above corresponds with the syntax for the expressions representing Cournot market output and price found earlier.

Once again, with the market price determined, it is possible to calculate firms' profits. Firm 1's Stackelberg profits, designated `pie1s`, is the firm's Stackelberg output, `Q1s`, multiplied by the difference between unit price and firm 1's unit cost `(Pstack - c1)`.

```
IN[]:=    pie1s = Simplify[Q1s*(Pstack - c1)]
```

$$OUT[]:= \frac{(a - 2\ c1 + c2)^2}{8\ b}$$

Similarly, firm 2's profits from playing a Stackelberg game, `pie2s`, are;

```
IN[]:=    pie2s = Simplify[Q2s*(Pstack - c2)]
```

$$OUT[]:= \frac{(a + 2\ c1 - 3\ c2)^2}{16\ b}$$

2. Comparison of Cournot and Stackelberg Models

In this final section we look at a Mathematica program that considers alternative oligopoly models and asks how our model specification might affect our predicted solutions. The point of this experiment is not only to illustrate that different models will generally lead to different solutions but to impress upon the reader the importance of choosing the correct model for the industry. Does the industry you want to model have a dominant firm that appears to lead the industry? If so, it might be more appropriate to model this industry as a Stackelberg oligopoly rather than a Cournot oligopoly. Do the firms in the industry produce a nearly homogenous good? If not, neither the Cournot nor the Stackelberg models will likely be an appropriate choice for modeling.

For the purposes of this experiment, we look at a duopoly industry with the same linear demand curve as in the previous programs. The only variation is that in the current model we make an additional simplifying assumption. In addition to our previous assumption that firms have constant marginal costs of production, we now assume that these costs are the same for both firms ($c_1 = c_2 = c$). Because of this assumption, the reader should be warned that our purpose is not to propose any quantitative differences between the models (although some qualitative differences will become apparent).

Rather our purpose is to drive home the message that modeling oligopoly markets is an art and a science. Modeling is the science. Choosing the right model is the art.

We investigate this special case of symmetric costs (i.e. marginal costs are the same constant number for all firms) by considering three alternative models' solutions. Our benchmark solution will be the collusive monopoly outcome. Unlike the non-cooperative Cournot and Stackelberg games, the collusive outcome assumes that the two firms agree to behave as a monopoly industry by restricting market output to the monopoly level with each firm producing half of the monopoly market output thereby splitting monopoly market profits. We use this as our benchmark because it is the most profitable possible outcome. We then return to our now familiar models of Cournot and Stackelberg Competition and ask: How do the predictions from these models differ from each other and how different are they from the collusive outcome?

In order to account for the collusive outcome, we take a slightly different approach to the general setup for this program than in the other Mathematica programs.

```
In[]:=      SetAttributes[a, Constant]
SetAttributes[b, Constant]
SetAttributes[c, Constant]

In[]:=      Clear[a,b,c,temp1,temp2];
Clear[Q,Q1,Q2,eqPr1,eqPr2,focPr1,focPr2,R2,pie1c,pie2c];

In[]:=      Price = a - b*(Q) ;
```

Notice that the setup still contains the `SetAttributes` commands, the `Clear` commands to clear any previously stored values, and our specification of our inverse demand relationship. Noticeably absent, however, is the definition of market quantity as the sum of firm 1's output and firm 2's output ($Q = Q_1 + Q_2$). This omission is intentional because we first want to consider the collusive (monopoly) outcome. This requires that we model the industry not as a duopoly, but rather as a monopoly and then divide the monopoly outcome between the firms.

As noted above, solving for the collusive outcome amounts to solving for the monopoly market output and then distributing this output evenly between the firms. To accomplish this, we develop an expression for the monopolists profit function.

```
IN[]:=    eqPrM = ProfitM == Q(P[Q] - c) ;

          eqPrM = Expand[%/. P[Q] -> Price]

OUT[]:=   ProfitM == Q (a - c - b Q)
```

Differentiate the profit function with respect to the choice variable, quantity (Q).

```
IN[]:=    focPrM = D[eqPrM, Q]

OUT[]:=   0 == a - c - 2 b Q
```

Then solve for the profit maximizing market quantity and name this quantity Qm,.

```
IN[]:=    Qm = Simplify[Q/. Solve[focPrM,Q][[1]]]

OUT[]:=   (a - c)/(2 b)
```

Because we assume that this collusive market output is distributed evenly between the duopolists, simply divide the market output, Qm., by two in order to find each firm's output.

```
IN[]:=    Q1m = Qm/2
OUT[]:=   (a - c)/(4 b)

IN[]:=    Q2m = Qm/2
OUT[]:=   (a - c)/(4 b)
```

We complete the Collusive Solution section of the code by calculating the collusive market price, `Pmon`, and the collusive profits of one of the representative firms (`pie1m`). Because both firms are assumed to split the market evenly, profits will be equal between the two firms.

```
IN[]:=    Pmon = Simplify[Price /. Q -> Qm]
OUT[]:=   (a + c)/2
```

```
IN[]:=    pie1m = Simplify[Q1m*(Pmon - c)]
OUT[]:=   (a - c)^2/(8 b)
```

Before continuing on to the Cournot and Stackelberg models we need to insert the definition for total market quantity that we had earlier omitted;

```
IN[]:=    Q = Q1 + Q2 ;
```

The next two sections of code model the Cournot Game and then the Stackelberg Duopoly Game. These models and their associated code are nearly identical to the Cournot and Stackelberg models considered earlier with the noted exception that the constant marginal cost of production is now the same for both firms (and is denoted `c` in the code).

In the last section of this code we take a closer look at the solutions to the models of our symmetric cost industry and make some analytic and graphic comparisons between the collusive, the Cournot and the Stackelberg outcomes.

The first set of comparisons that we make considers the overall size of the market that the alternative models predict. The `N` in the first input statement below is a Mathematica operator to return a numerical value. Also by the precedence rules the term in the first input statement below is (Qcour/Qm) – 1

```
IN[]:=    sizeQc = N[(Qcour/Qm-1)*100]
OUT[]:=   33.3333
```

```
IN[]:=    sizeQs = N[(Qstack/Qm-1)*100]
Out[]:=   50.
```

The first statement above compares the size of the Cournot market output in our symmetric cost model with the collusive outcome and shows that in the Cournot model output is 33.3% larger than the most profitable collusive market size. The second input statement above makes a similar comparison between the Stackelberg market output and the collusive market size and shows that the Stackelberg market output is even larger – 50% larger than the collusive market size.

The next set of statements compares the size of the individual firms' outputs that the alternative models predict.

```
IN[]:=      sizeQ1c = N[(Q1c/Q1m-1)*100]
OUT[]:=     33.3333

IN[]:=      sizeQ2c = N[(Q2c/Q2m-1)*100]
OUT[]:=     33.3333

IN[]:=      sizeQ1s = N[(Q1s/Q1m-1)*100]
OUT[]:=     100.

IN[]:=      sizeQ2s = N[(Q2s/Q2m-1)*100]
OUT[]:=     0
```

The first two input statements above compare firm 1's and firm 2's market output from the Cournot game against their collusive output levels and show that both produce 33.3% more in the Cournot model than the collusive outcome. The last two input statements making similar comparisons of the firms' market outputs from the Stackelberg game against their collusive output. In our symmetric costs industry we find that the Stackelberg leader (firm 1) produces twice the collusive output and the Stackelberg follower (firm 2) produces the same output as in the collusive game.

At this point we have shown that the collusive (monopoly) outcome has the smallest market size and therefore the highest market price. Similarly it is now clear that the Stackelberg model has the largest market size and consequently the lowest market price. What is still unclear is how industry and firms' profits compare. The first set of Mathematica statements below compare Cournot profits to the collusive shared monopoly profits.

```
IN[]:=      sizePie1c = N[(pie1c/pie1m-1)*100]
OUT[]:=     -11.1111
IN[]:=      sizePie2c = N[(pie2c/pie2m-1)*100]
OUT[]:=     -11.1111
```

Here we find that the Cournot profits for both firms are 11.1% lower than the profits that would be achieved if the firms were to collude by acting as a monopolist.

The next set of statements compares the Stackelberg profits to collusive profits.

```
IN[]:=      sizePies = N[((pie1s + pie2s)/(2*pie1m)-1)*100]
OUT[]:=     -25.
IN[]:=      sizePie1s = N[(pie1s/pie1m-1)*100]
OUT[]:=     0
IN[]:=      sizePie2s = N[(pie2s/pie2m-1)*100]
OUT[]:=     -50.
```

The first statistic (that we call `sizePies`) compares industry profits of the Stackelberg and collusive outcomes. The other input statements compare the profits between these alternative outcomes for firm 1 and firm 2 respectively. Notice that industry profits for this Stackelberg game are 25% lower than the monopoly profit level for the industry. Therefore, industry profits are lower in the Stackelberg game than they were in the Cournot game. However, despite having less industry profits in the Stackelberg game, we find that the Stackelberg leader (firm 1) is able to achieve higher profits in than in the Cournot game. In fact, in this symmetric cost industry, we find that the Stackelberg leader earns the same profits as in the collusive outcome.

Given that collusive and Stackelberg profits are the same for firm 1, why doesn't the Stackelberg leader produce at half the monopoly output? Because in such a one-shot game, firm 2 would cheat and produce more than its collusive share. It is left to the

reader to show what quantity firm 2 would produce after observing firm 1 produce half the monopoly market output.

The results presented above are also summarized in the Mathematica program with a series of pie charts.

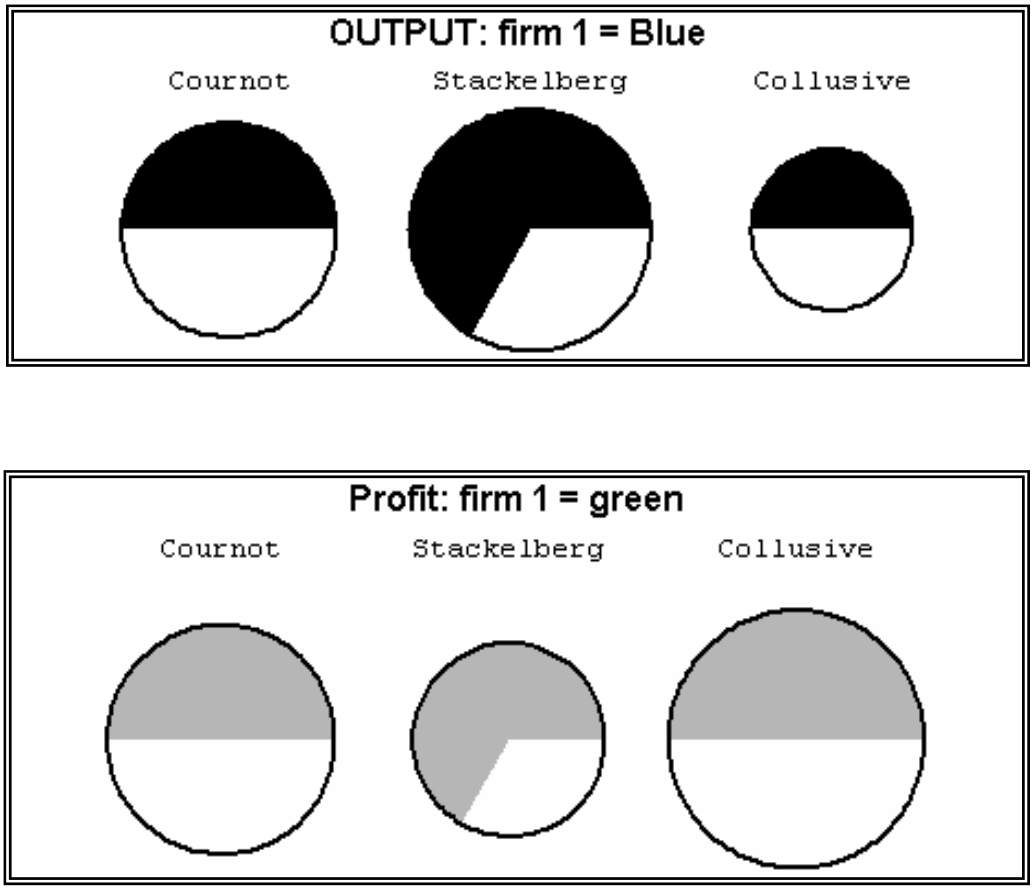


Figure 10.1 Pie Charts for Output and Profit

The Mathematica code that generates these sets of pie charts and the plots that follow is longer and more complicated than the code statements used throughout this paper and therefore omitted here to avoid confusing the reader. However, all of the code is presented in the files and the adventurous reader is encouraged to study it with the aid of a Mathematica reference manual.

The program `combine.nb` concludes with a series of plots illustrating the solutions to the Cournot and Stackelberg game. The first of these plots (Fig. 10.2) represents the firms' isoprofit curves. This is a contour mapping (similar to a geographic map) that shows how the two firms' profits vary for different combinations of output by the two firms.

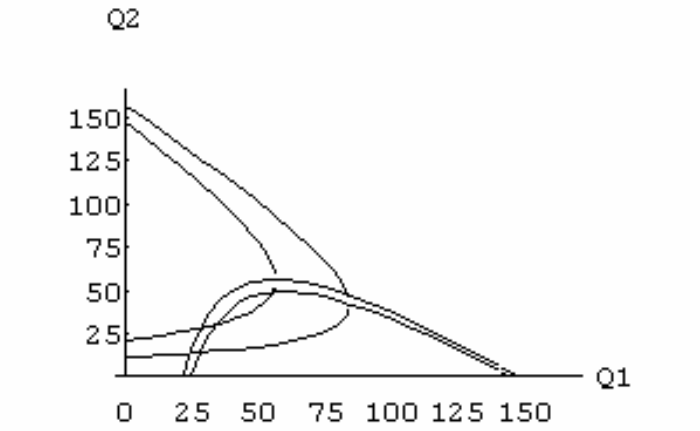


Figure 10.2 Contour Maps of Two Firms' Profits

Each isoprofit curve represents a different level of profits and the level of profits increase as the curves shift towards their respective axis. In the plot above, firm 2's isoprofit curves are red and firm 1's isoprofit curves are blue.

In the final plot, the isoprofit curves are combined with the plot of the reaction functions that we considered earlier.

Figure 10.3 Isoprofit Curves Combined with Reaction Function

3. Experiments

In this chapter we develop a series of experiments that cover many of the aspects of the models presented. However, one set of experiments is to consider modeling alternative market structures.

Chapter 11

Genetic Algorithms and Evolutionary Games in MATLAB

Genetic algorithms are search procedures based on the logic of natural selection and genetics. The central concept of genetic algorithms is "survival". A group of individuals - each one represented, for their computational implementation, by a string of characters, usually based on a binary code - compete with one another and the "most fit" survives to give birth to a next generation of related individuals. This process continues through a number of generations leaving at the end the "most fit" individual.

One application of genetic algorithms is to evolutionary game theory. In this chapter we present an example from this field, namely an iterated prisoner's dilemma problem.[28] First we illustrate some basic concepts of genetic algorithms and evolutionary games using very simple examples. Next we show how to work with binary representations in MATLAB. Then we present a basic MATLAB program and perform some experiments. A more sophisticated MATLAB program of genetic algorithms which builds on the one in this chapter will be presented later in the Genetic Algorithms and Portfolio Model in MATLAB chapter.

1. Introduction to Genetic Algorithms

There are different types of genetic algorithms. Here we will introduce one of the most commonly used. The algorithm starts with the generation of the initial population, and then we have a repetitive process that evaluates the fitness, selects, crosses, mutates and replaces the old population. This process will be stopped when we reach the required precision after a number of generations. In this section we will present a simple example. We will later suggest ways of introducing more complex procedures.

[28] The use of this example was initially motivated by the work of our student Shyam Gouri Suresh.

Let's assume that we start from a given initial population of only four individuals. Each individual's characteristics are represented by a string of five binary digits (chromosomes).

Initial population

1) 00000
2) 10101
3) 11010
4) 11100

We are all used to dealing with decimal representation of numbers. For example, a number like 142 (one hundred and forty two) is constructed in the following way

$$(1)\ 10^2 + (4)\ 10^1 + (2)\ 10^0 =$$
$$100 + 4 + 2 = 142$$

A binary representation works in a similar way, but with a different base: two instead of ten. Thus, a string of characters such as the second individual in the initial population above, i.e. 10101, can be interpreted as representing the number 21 (twenty one), since

$$(1)\ 2^4 + (0)\ 2^3 + (1)\ 2^2 + (0)\ 2^1 + (1)\ 2^0 =$$
$$16 + 0 + 4 + 0 + 1 = 21$$

The usefulness of this kind of representation will be appreciated soon in the crossover and mutation steps.

Given the initial population, we need some fitness criteria in order to select the "best" individuals. This criterion depends on the specific problem under consideration, and it is usually represented by a mathematical function to be applied to each individual. For the sake of simplicity, let's assume here that the fittest individuals will be those with the highest numerical values associated with their characteristics (their "chromosomes"). Thus, the third and fourth individuals in the initial population above would be selected for reproduction, and they would form a couple. This couple will have four children which will replace the entire previous generation of individuals. Each new individual will be generated in the following way: first there will be a crossover of the last two genes in the strings of the selected couple as follows

Couple	Crossover
3) 11010	110**00**
4) 11100	111**10**

and then there will be a mutation of the last gene for each of the crossover results. That is

Crossover	Mutation
110**00**	1100**0**
	1100**1**
111**10**	1111**0**
	1111**1**

Thus, from the mutation column above, the second generation ordered in an ascendant way according to numerical value, will be

Generation 2

1) 11000
2) 11001
3) 11110
4) 11111

We now select again the two fittest individuals, obviously the third and fourth. Then we can again apply the same crossover and mutation procedures. The result is shown below.

Couple	Crossover	Mutation
		1111**1**
3) 11110	111**11**	1111**0**
4) 11111	111**10**	1111**0**
		1111**1**

Thus, the third generation, ordered in ascendant way according to numerical value, is

Generation 3

1) 11110
2) 11110
3) 11111
4) 11111

Observe that we have reached the highest possible values for the third and fourth individuals, which will be selected as the parents of the next generation. Actually, if we repeat the crossover, mutation and selection steps from now on, we will find that all the next generations will be identical to generation 3. Thus, we can conclude that we have reached an optimum, that is, the fittest individual given the characteristics of our problem.

The example presented above is simple; yet it provides a basis from which we can introduce several modifications to have an idea of what the actual practice in the field of genetic algorithms is like. For example, the size of the population could be larger or the string of characters corresponding to each individual could be longer. The initial population could be generated stochastically. The fitness criteria, as we mentioned above, could be of a different nature from the one used here, and represented by a specific fitness function. Given a larger population, more than one couple could be selected to be the parents of the next generation. To this end, a stochastic procedure could be used to form couples out of a pool, also determined with some degree of randomness, of the fittest individuals. The crossover point - the last two genes in our example - could also be randomly determined at each generation, as well as the mutating genes, which we arbitrarily chose to be only and always the last one.

Before we introduce the code for our model, we will present a simple example of an evolutionary game and later introduce a number of MATLAB functions that can be used for manipulating binary representations.

2. A Simple Example of Evolutionary Game

An evolutionary game is a game in which strategies evolve through a process of dynamic selection. As an example, we will present here a simple version of the game known as iterated prisoner's dilemma. The prisoner's dilemma was introduced and analyzed earlier in the Cournot Duopoly in Mathematica chapter. Our game will have the representation shown in Table 11.1, where D means defect and C means cooperate.

		Player II	
		D	**C**
Player I	**D**	1 , 1	5 , 0
	C	0 , 5	3 , 3

Table 11.1 Game Representation

Thus if the two individuals cooperate with one another they will each receive a gain of three but if they both defect they will each receive a gain of only one. In contrast, if Player I decides to cooperate but Player II defects, then Player I will make a gain of zero and Player II will make a gain of five.

We will assume that this game will be played many times by successive generations of individuals. Each individual will be represented by a chromosome of 24 bit length. Each gene of the chromosome will represent an action (0 for defect, 1 for cooperate). Thus, each individual chromosome will be interpreted as a strategy, which is a sequence of actions.

Within a given generation, each individual will play 24 times against each member of her generation following the strategy implied by her chromosomes, and her resulting payoffs will be accumulated. At the end of each generation round, the two individuals with the highest accumulated payoffs will be selected to be the parents of the next generation. Children will be born out of the crossover and mutation of parents' chromosomes. The process will be repeated for a number of generations.

Notice that, given the simple formulation of this iterated game, individuals will not think and act strategically. They will just follow their strategies regardless of their opponent's actions. In this sense they are very stubborn and simple-minded agents.

However, we know that the most efficient individual action for the prisoner's dilemma game is to defect, and that the unique Nash equilibrium is (D,D). The question for our experiments is: Would this population of simple-minded agents evolve in such a way that only defectors will survive? That is, will the selective evolution of the population generate an outcome similar to the one that would be reached by rational and strategic-thinking players which, by the way, implies that everybody will be worse-off than in the case in which everybody cooperates?

3. Working with Binary Representations in MATLAB

When using binary variables to code genetic algorithms, the key concept to keep in mind is that the variables can be specified as integers but can also be thought of as strings of binary variables. Then for example, the integer 25 would be represented in an 8-bit binary string as

00011001

that is as

$$\begin{aligned} &0\left(2^7\right)+0\left(2^6\right)+0\left(2^5\right)+1\left(2^4\right)+1\left(2^3\right)+0\left(2^2\right)+0\left(2^1\right)+1\left(2^0\right) \\ &=0(128)+0(64)+0(32)+1(16)+1(8)+0(4)+0(2)+1(1) \\ &=25 \end{aligned}$$

Thus we can create an integer variable in the program, say `genepool = 25 ;` and use that to represent both a player's strategy (or, as we will see in the Genetic Algorithms and Portfolio Models chapter, an economic variable such as the percentage weight of an asset in a portfolio) and at the same time manipulate it as a bit string in a genetic algorithm code.

MATLAB provides a variety of functions to manipulate the binary representations of numbers. The functions `dec2bin`, `bitor`, `bitand`, `bitshift` and `bitcmp` are used in the genetic algorithm code in this chapter and in the Genetic Algorithms and Portfolio Models chapter.

The function `dec2bin` converts a decimal integer to a binary string. For example, the statement

```
x = dec2bin(6);
```

returns the binary string 110 which corresponds to the decimal number six, while

```
x = dec2bin(6,8);
```

returns a binary representation of the decimal number six with at least eight characters, that is

00000110.

The function `bitor` returns the decimal bit-wise OR of two nonnegative integer numbers. That is, it compares bit-to-bit each binary position of the two numbers generating a 1 whenever it finds the combination (1,1), (1,0) or (0,1) and generating a 0 when it finds the combination (0,0). For example, the 4-bit representation of the numbers 7 and 9 are respectively

0111 and 1001

and the bitwise OR operation on these numbers yields 1111, which corresponds to the decimal representation 15. Thus the statement

```
x = bitor(7,9);
```

returns the number 15.

The function `bitand` returns the bit-wise AND of two nonnegative integer numbers. Comparing bit-to-bit the binary representation of two numbers generates a 0 whenever it finds the combination (0,0) or (0,1) or (1,0) and generates a 1 when it finds the combination (1,1). Thus the statement

```
x = bitand(7,9);
```

returns the decimal number 1.

The function `bitshift`

```
x = bitshift(A,k);
```

returns the value of the nonnegative integer A shifted by k bits (to the left when k is positive, to the right when k is negative and filled with zeroes in the new spaces). For example, when A = 14 its binary representation is 01110. Thus the statement

```
x = bitshift(14,1);
```

corresponds to the binary representation 11100 and returns the decimal value 28. If the shift causes x to overflow, the overflowing bits are dropped.

Finally, the function `bitcmp`

```
x = bitcmp(A,n);
```

returns the bit complement of A in the form of an n-bit floating point integer, i.e. each 0 is replaced with a 1 and vice versa. Thus, the statement

```
x = bitcmp(28,5);
```

where 28 is represented with 5 binary digits as 11100 and its complement is 00011 and thus the function returns the decimal value 3.

4. Overview of the MATLAB Code

There is a Genetic Algorithm and Direct Search Toolbox for use with MATLAB which includes routines for solving optimization problems using genetic algorithms, and where the user can directly specify some overall characteristics of his or her problem (i.e. population size, fitness criteria, number of generations, etc.) without having to pay special attention to the workings and implementation of the genetic algorithm. However, to provide an opportunity to learn some basic concepts about genetic algorithms and to go deeper into learning the MATLAB software we introduced earlier in the Portfolio Model chapter, we are basing this chapter on a genetic algorithm code initially developed by one of our students, Huber Salas.

While in an earlier chapter we introduced relatively simple MATLAB programs, here we advance to a program that calls a number of built-in MATLAB functions and uses a number of M-files. MATLAB has two kinds of M-files that can be written by users: (1) *scripts* which do not accept input arguments and (2) *functions* that do accept input arguments.[29] They contain a series of statements and can be stored in an independent MATLAB file. The *functions* receive a number of input variables, process them, and return one or more output variables. The name of the main program we will present here is `gagame.m`, i.e. it is a genetic algorithm evolutionary game problem. This program and all the functions it calls are available in the book web page.

The basic structure of the program, shown below, consists of three main parts. The first part contains the initialization of counters and parameters and a function call to initialize the population. The second part is a `for` loop across generations that in turn contains several function calls. Finally, the third part contains commands to print and graph the main results.

[29] For a discussion of MATLAB M-files see the manual "Getting Started with MATLAB" in the MATLAB Help menu options.

```
% initialization of counters and parameters;
nruns = 100;   popsize = 8;
clen = 24;     pmut = 0.5;

% generation of chromosome strings of initial population
genepool= initpoprand_gagame(popsize);

for k = 1:nruns;

      % computation of fitness function and fittest individual
      [fit, bestind, bestfit] = fitness_gagame(genepool,popsize,clen);
      wbest(k) = bestind;
      fbest(k) = bestfit;

      % selection of parents;
      [parent0,parent1] = parentsdet(fit,genepool);

      % crossover of parents chromosome strings
      [child0,child1] = crossover(clen,parent0,parent1);

      % mutation of children chromosome strings
      for h = 1:2:popsize;
          child0mut = mutation(pmut,clen,child0);
          genenew(h) = child0mut;
          child1mut = mutation(pmut,clen,child1);
          genenew(h+1) = child1mut;
      end
    genepool = genenew;
end

% print and graph fittest individual;
dec2bin(wbest(nruns),clen)
fbest = fbest / (clen * (popsize - 1));

figure(1);
xaxis = [1:1:nruns]';
plot(xaxis,wbest);
figure(2);
xaxis = [1:1:nruns]';
plot(xaxis,fbest);
```

In the initialization of counters and parameters section we set the number of runs `nruns` and the population size `popsize`. We also set the length of the chromosome string `clen` and the probability of a child mutation `pmut`.

We then call the function `inipoprand_gagame` to initialize the vector

`genepool` which will contain a number of individuals equal to the population size, each individual represented by a 24-bit chromosome string. So, in our example, `genepool` is a vector of 24 bit chromosomes with an element for each of the eight individuals in the population.

Then we move on to the main `for` loop in the program, running from 1 to the number of runs. This loop contains a sequence of function calls. It starts with a call to the `fitness_gagame` function to compute the fitness function for each individual and to select the fittest individual, which at each run will be stored in the `k`th element of the vector `wbest` while the corresponding criterion value will be stored in the `k`th element of the vector `fbest`. Thus, at the end of the runs, these vectors will contain the sequence of optimal chromosome strings and optimal criterion values respectively.

Next the function `parentsdet`, using the fitness function previously computed, will select two parents (`parent0` and `parent1`) who will form a couple. This is followed by a call to the function `crossover` which will generate two children (`child0` and `child1`) as the product of the crossover of the chromosome strings of the two parents.

Next comes a `for` loop whose index goes from 1 to `popsize` in increments of two. In this loop, out of the two newborn children a new generation will be created through mutations of their chromosomes. Half of the new generation will come out of mutations of the first child (`child0`) and the other half will come out of mutations of the second child (`child1`). At every pass through the `h` loop the function `mutation` is called twice. This has the effect of generating two mutated children, and their 24-bit chromosome representations are stored in subsequent cells of the `genenew` vector.

Once the new generation is created, the new vector `genenew` replaces the old vector `genepool` and the main loop of the program starts over again.

After the main loop goes through the established number of runs, the statement

```
dec2bin(wbest(nruns),clen)
```

prints the last element of the vector `wbest` which contains the chromosome string of the fittest individual. Then the statement

```
fbest = fbest / (clen * (popsize - 1));
```

is used to compute the average value of the optimal criterion value at each run. Notice here that `fbest` is a vector and `(clen * (popsize - 1))` is a scalar so that the division operation is repeated for each element in the vector. Finally, the vector of fittest individuals `wbest` and the vector of optimal criterion values `fbest` are plotted.

This provides an overview of the program. It is important to point out that every time you run the program, particularly when changing the number of generations or the population size, you should clean out the old commands and workspace to avoid displaying spurious results. To do so, go to Edit in the top MATLAB menu. Then select Clear Command Window and confirm with Yes that you want to do this. Then do the same for Clear Command History and for Clear Workspace.

We will now present each function in detail.

5. Functions

5.1 Initpoprand_gagame

This function is simple in that all it does is to assign a random number to each individual string of chromosomes. Thus the MATLAB code for this initialization function is

```
function genepool= initpoprand_gagame(popsize,clen);
for k1 = 1:popsize;
  genepool(k1) = ceil(rand * (2^clen)-1)));
  dec2bin(genepool(k1), clen)
end
```

The header statement for the function, i.e.

```
function genepool = initpoprand_gagame(popsize,clen);
```

tells us that the name of the function is `initpoprand_gagame`, that the arguments

`popsize` and `clen` will be passed to the function and the result `genepool` will be returned by the function.

Then a loop going from 1 to `popsize` is used to assign a random value to each element of the `genepool` vector. The statement

```
genepool(k1) = ceil(rand * (2^clen)-1)));
```

assigns to each element of the vector the ceiling (the nearest higher integer value) of the result of multiplying the variable `rand` (a zero-one uniform distribution random number generator) times the number `(2^clen)-1`. This last number will be equal to the highest possible value represented with a binary string of length equal to `clen`. For example, if `clen` equals three, that number will be equal to two to the power of three, i.e. eight, minus one. Thus the number is seven, whose binary representation is 111.

The statement

```
dec2bin(genepool(k1),clen)
```

does not play an essential role in the function since it only serves to print a binary representation of `genepool` for debugging purposes. Since there is no semicolon at the end, this statement will return and print the 24-bit binary representation of each element of `genepool`. Finally, notice that an `end` statement is not necessary at the end of a MATLAB function, unlike the cases of `for` loops or conditional `if` statements.

5.2 Fitness_gagame

This `fitness_gagame` function contains the game to be played and a procedure to select the fittest individual which is similar to the one used in the portfolio chapter earlier in the book. The first part of the function consists of three nested loops: the first one for player1, the second one for player2, and the third one for games. Thus, each player selected in the first loop will play against each other player selected in the second loop. These two players will play 24 games, playing in each game the action determined by the corresponding gene in their chromosome sequence. The statements for the first part of the function are shown below.

```
function [fit,bestind,bestfit] = fitness_gagame(genepool,popsize,clen);
payoffs(1,popsize) = 0;
% Loop for player1
for k1 = 1:popsize;
    strategyp1 = genepool(k1);

    % Loop for opponents (player2)
    for k2 = 1:popsize;
    strategyp2 = genepool(k2);

        if (k1 ~= k2)
            mask = 1;

            %Loop for games
            for k3 = 1:clen;
            actionp1 = bitand(strategyp1,mask);
            actionp2 = bitand(strategyp2,mask);
            mask = bitshift(mask,1);
                % defect, defect
                if (actionp1 == 0) & (actionp2 == 0)
                    payoffs(k1) = payoffs(k1) + 1;
                end
                % cooperate, defect
                if (actionp1 > 0) & (actionp2 == 0)
                    payoffs(k1) = payoffs(k1) + 0;
                end
                % defect, cooperate
                if (actionp1 == 0) & (actionp2 > 0)
                    payoffs(k1) = payoffs(k1) + 5;
                end
                % cooperate, cooperate
                if (actionp1 > 0) & (actionp2 > 0)
                    payoffs(k1) = payoffs(k1) + 3;
                end
            end % end loop games

        end % end if

    end % end loop opponents

end % end loop player1
```

The function begins with the statement

```
payoffs(1,popsize) = 0;
```

which initializes to zero a vector that will contain the accumulated payoffs of player1.

This is followed by the beginning of the `k1` loop for player 1. The statement at the beginning of this loop

```
strategyp1 = genepool(k1);
```

and the statement at the beginning of the loop for player 2

```
strategyp2 = genepool(k2);
```

assign to the temporary variables `strategyp1` and `strategyp2` the chromosomes of player 1 and player 2 respectively, that is the strategies each player will play.

Next, at the beginning of the loop for games, the statements

```
actionp1 = bitand(strategyp1,mask);
actionp2 = bitand(strategyp2,mask);
```

select the actions to be played at each game out the strategies of each player. The variable `mask` was previously initialized with the value 1. Thus, its 24-bit binary representation will be

00000000 00000000 00000001

Remember that the function `bitand` returns the bit-wise AND of two nonnegative integer numbers. Thus, comparing bit-to-bit the binary representation of two numbers, it generates a 0 whenever it finds the combination (0,0) or (0,1) or (1,0) and generates a 1 when it finds the combination (1,1). Thus the temporary variables `actionp1` and `actionp2` will contain the first gene of each player's chromosome, that is the first action to be played in the first game. For example, if the chromosome of player 1 is

01010101 11111111 00001111

the result of the `bitand` operation will be

00000000 00000000 00000001

and the content of the `actionp1` variable will be the number 1, so that the player will cooperate. The statement

```
mask = bitshift(mask,1);
```

shifts the mask one position to the left, resulting in

00000000 00000000 00000010

Thus, at each pass of the loop, the number 1 will shift one position to the left so that the next action will be selected. The remaining of the loop for games, i.e. the `k3` loop, accumulates the payoffs for player 1 depending on the result of the game. Each of the four possible outcomes is evaluated. For example, in the case player 1 plays 0 (defect) and player 2 plays 1 (cooperate) the sentences

```
% defect, cooperate
if (actionp1 == 0) & (actionp2 > 0)
payoffs(k1) = payoffs(k1) + 5;
```

add 5 to the element of the `payoffs` vector corresponding to the game being played.

Once the main loop of the function - the loop for player 1 - is completed for all players, the second part of the function selects the fittest individual in the generation, which is the one with the highest payoffs. It begins with the statement

```
fit = payoffs;
```

to assign the variable `payoffs` to the temporary variable `fit`. The statement

```
[top topi] = max(fit);
```

then returns the value (`top`) and the index (`topi`) corresponding to the maximum value in the fitness vector `fit`. Finally, the `topi` index is used to assign to the "best individual" vector, `bestind`, the corresponding chromosome with the statement

```
bestind = genepool(topi);
```

while the corresponding value of the criterion function vector is assigned to the variable `bestfit` with the statement

```
bestfit = top;
```

With the fitness now determined we turn next to the selection of parents.

5.3 Parentsdet

The `parentsdet` (parents deterministic) function is a simple deterministic function that selects the two individuals that will be the parents of a new generation. The simple selection method used in the present function will be to select the two individuals with the highest criterion value or fitness.

```
function [parent0,parent1] = parentsdet(fit,genepool);
[top topi] = max(fit);
parent0 = genepool(topi);
fit(topi) = 0;
[top topi] = max(fit);
parent1 = genepool(topi);
```

As in the previous function, the statement

```
[top topi] = max(fit);
```

returns the index `topi` corresponding to the maximum value of the `fit` vector that is the fittest individual. Using that index, the corresponding chromosome string of the first parent is stored in the variable `parent0`. To select the second parent, we set to zero the

fitness value of the previous maximum and proceed in the same manner as before now to select the second parent (`parent1`). We will see later, in the chapter Genetic Algorithms and Portfolio Model in MATLAB, in the `parentsrand` function, how to implement a more sophisticated random procedure for parent selection in which more fit parents have a "higher chance" of generating off springs.

5.4 Crossover

The crossover mixes the chromosome information of the two parents to create two children. We will consider here only the case of a single crossover. The function code is shown below.

```
function [child0,child1] = crossover(clen,parent0,parent1);
crossov = ceil(rand*(clen));
maska = 1;
for k = 1:(crossov-1)
    maska = bitshift(maska,1);
    maska = maska + 1;
end
child0 = bitor (bitand(parent0, maska),
    bitand(parent1,bitcmp(maska,clen)));
child1 = bitor (bitand(parent1, maska), bitand(parent0,
    bitcmp(maska,clen)));
```

To determine the crossover point we use the statement

```
crossov = ceil(rand *(clen));
```

Remember that the `clen` variable contains the chromosome length of the individual (24 bits). Then, multiplying `clen` times the uniform zero-one random number generator function `rand` we are randomly choosing the crossover point. Since that point has to be an integer number, we apply the function `ceil` to the result which rounds off the result to the nearest higher integer.

Next we assign the initial value 1 to the mask variable `maska`.

00000000 00000000 00000001

Then we pass through a loop that goes from 1 to the crossover point. At

each pass, we shift the 1 one position to the left

00000000 00000000 00000010

and we add the value one to the result, thus switching the rightmost bit from zero to one.

00000000 00000000 00000011

Thus if the crossover point was 12, we will end up with the mask

00000000 00001111 11111111

Next we generate the first child with the statement

```
child0 = bitor (bitand(parent0, maska),
                bitand(parent1, bitcmp(maska,clen)));
```

For example, consider the case where `parent0` has the chromosome string

00010001 00010001 00010001

In this case the statement

```
bitand(parent0, maska)
```

would apply the mask

00000000 00001111 11111111

and create the chromosome string

00000000 00000001 00010001

Also assume that `parent1` has the following chromosome string

10001000 10001000 10001000

Thus the statement

```
bitand(parent1, bitcmp(maska,clen)));
```

would apply the complement of `maska`, that is

11111111 11110000 00000000

to `parent1` with the `bitand` operation to obtain

10001000 10000000 00000000

Finally, the application of the `bitor` function to the chromosome strings

00000000 00000001 00010001
10001000 10000000 00000000

will generate the result

10001000 10000001 00010001

which will be the chromosome string of the first child (`child0`). The second child is obtained in a similar fashion, reversing the position of the parents in the corresponding statement.

5.5 Mutation

The mutation function generates a random mutation in a single bit of the chromosome string of a child. The code is shown below.

```
function f = mutation(pmut,clen,child)
tt = 1;
if (rand < pmut)
    idx = round(rand*(clen-1));
    tt = bitshift(tt,idx);
    temp = bitand(child,bitcmp(tt,clen));
```

```
    if(temp==child)
        child = child + tt;
    else
        child = temp;
    end
end
f = child;
```

Recall that `pmut` is the probability of a child mutation. The `tt` variable is initially set to one and will be bit shifted to create the mutation at the desired point in the chromosome. The scalar integer `idx` is the index of the location that is one less than where the mutation will occur. It is determined by rounding off the randomly generated location using the zero-one uniform random variable `rand` and the length of the chromosome `clen` less one.

Consider for example a case where the index variable is set to three. Also, recall that the variable `tt` is set to one. Thus the `bitshift` function call

```
tt = bitshift(tt,idx);
```

shifts the `tt` binary variable three positions to the left so that it becomes

1000

and the mutation is going to be done in the fourth position. Then the mutation is done with the statement

```
temp = bitand(child,bitcmp(tt,clen));
```

and the result is stored it in the temp variable. Consider first just the `bitcmp` part of this statement. It yields the 24-bit complement to the `tt` variable, which is

11111111 11111111 11110111

Then the `bitand` operation is applied to this bit string and the `child` variable to obtain the mutated string, which is stored in the `temp` variable.

The `bitand` operation produces the desired mutation if the bit to be changed was a one. However, it does not produce the correct result if the bit to be changed was a zero. Therefore it is necessary to add the following lines of code

```
if(temp==child)
      child=child+tt;
else
      child=temp;
end
```

In the case where the bit to be changed was a zero the `bitand` operation above will have produced no change in the chromosome and it is necessary to accomplish that by adding an integer amount `tt` to the variable. In our case the binary representation of the `tt` variable was

1000

so its integer value is 16. Then when 16 is added to the `child` variable it produces the desired mutation by changing the zero bit in the fourth location to a one bit.

On the other hand if the `temp` variable is not equal to the `child` variable - as occurs when the bit to be changed is a one - then it is only necessary to set the `child` variable equal to the `temp` variable.

This completes the discussion of the mutation function and indeed the discussion of all the functions and leaves us free to turn our attention to the results obtained by using the program.

6. Results

Figure 11.1 below shows the results of running the main program `gagame.m` with a number of runs equal to 100 and a population size of 8, starting from a random initial population. The first graph shows the decimal representation of the chromosome of the fittest individual at each run. We can observe that after about ten runs this value converges to zero and stays there. This corresponds to the chromosome

00000000 00000000 00000000.

Thus, the optimal strategy that results from the simulation is to always defect. The second graph shows the evolution of the corresponding average payoffs. We can see how these payoffs converge to a value near one, which is the value corresponding to the Nash equilibrium of the game.

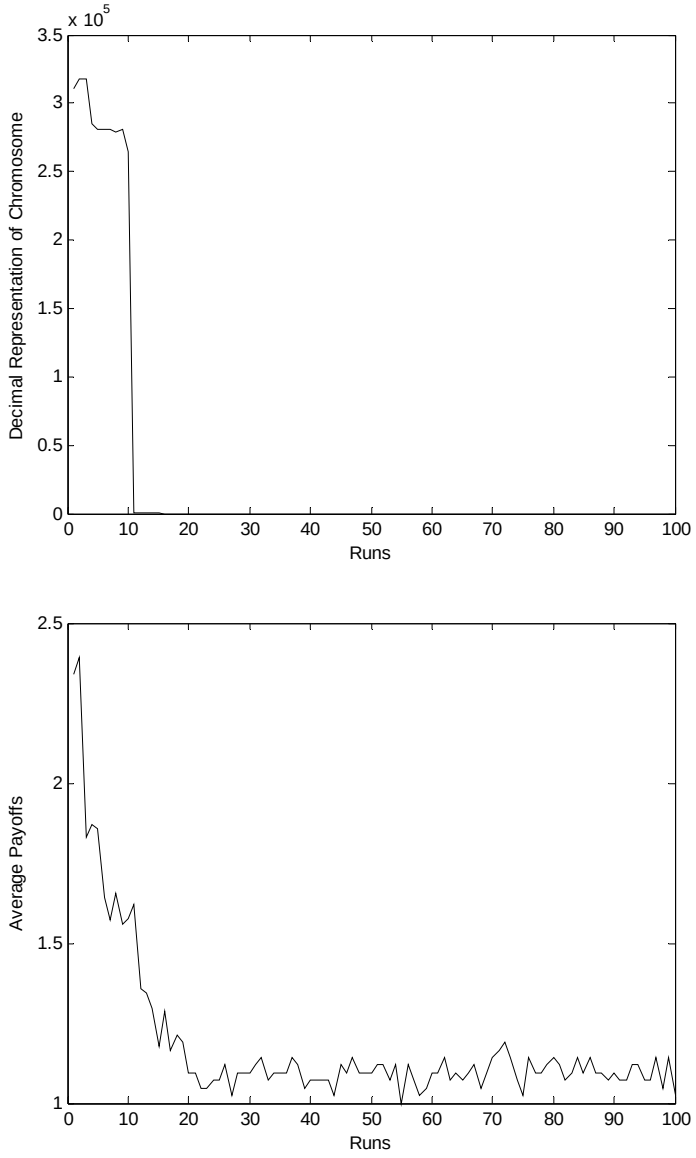


Figure 11.1 Evolutionary Game with Random Initial Population

Figure 11.2 shows the results of an experiment in which the initial population is composed entirely of cooperators. That is, individuals with a chromosome equal to

11111111 11111111 11111111

To run this experiment, in the `initpoprand.m` function we have to replace the statement

```
genepool(k1) = ceil(rand * (2^clen)-1);
```

with the statement

```
genepool(k1) =  (2^clen)-1;
```

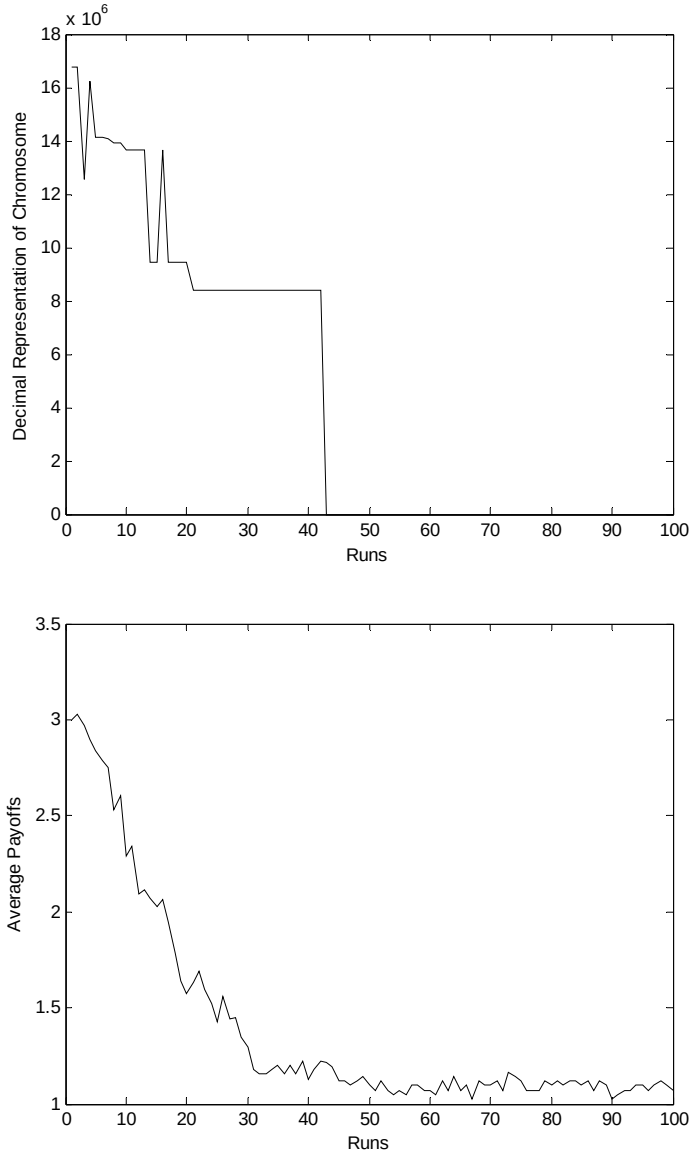


Figure 11.2 Evolutionary Game with Initial Population of Cooperators

We can see in Fig. 2 that the results converge, at a slower pace than in the previous experiments, to the same outcome. The fittest individuals will be defectors, born out of mutations and successive selections across generations. Interestingly, a population of all

cooperators, thus achieving the higher possible payoffs, when suffering even mild mutations such as the ones implied by our MATLB code, will end up transformed into one of all defectors with an inferior standard of living.

8. Experiments

The simplest experiments with this genetic algorithm would be to change the number of model iterations and/or population size to see how this affects the outcome. You may also want to try an experiment in which the initial population is composed of all defectors and see if they ever become all cooperators.

A more challenging set of experiments will be to introduce further refinements in the code to move closer to the actual practice in the field of genetic algorithms, such as the random selection of parents and the selection of more than one couple to be the parents of the next generation. Before doing so, you are encouraged to read the chapter Genetic Algorithms and Portfolio Model in MATLAB where these refinements are introduced.

More interesting experiments that would get you closer to the practice in the filed of evolutionary games involve some sort of strategic thinking and behavior on the part of players. Instead of being taken regardless of the opponent's actions, a player's actions will be determined for example as reactions to the opponent past behavior (Axelrod (1997)). Also it would be interesting to explore the evolutionary dynamics of a spatial model of local interaction in which each individual plays the prisoner's dilemma with her neighbors (Nowak and May (1992) and (1993)).

In both these cases - more sophisticated strategies or local interaction - it is found that the evolutionary behavior differs from the convergence to all defectors we found in this chapter. Indeed, it is usually the case that the evolution converges to cooperation or even displays complex patterns of cyclical behavior.

Since the MATLAB representation of these models may be more demanding than the one presented in this chapter, before moving in this direction you are encouraged to read the chapter on Agent-Based Models in MATLAB to learn about more sophisticated modeling techniques which may be useful to program problems of this nature.

8. Further Reading

A classic reference in the genetic algorithms literature is Goldberg (1989). For introductions to evolutionary games see the Stanford Encyclopedia of Philosophy (2005) and Axelrod (1997).

Chapter 12

Genetic Algorithms and Portfolio Models in MATLAB

In this chapter we present an example that builds on the Markowitz optimal portfolio model we used earlier in the Portfolio Model in MATLAB chapter. In that chapter we used two different methods to solve the Markowitz problem: first a Monte Carlo optimization search method then a MATLAB gradient optimization function. Here we will use a genetic algorithm based on the one we presented earlier in the Genetic Algorithms and Evolutionary Games in MATLAB chapter.

First we solve the same convex problem we solved in the Portfolio Model chapter. It has a unique global maximum - given the quadratic nature of the criterion function to be optimized. Later in this chapter we will introduce a more difficult but more realistic problem by means of including brokerage fees which may result in non-convexities and thus in a number of local maxima. It is for this kind of problems that genetic algorithms are particularly useful since they are global optimization algorithms. They perform a global exploration of the optimization space and are less likely to be trapped by local minima or maxima than is the case for other standard optimization procedures.

1. Overview of the MATLAB Code

Remember that the Markowitz problem was stated in an earlier chapter as to find x to maximize J in

(1) $$J = \mu' x - \frac{1}{2}\beta x' \Sigma x$$

subject to the constraints

(2) $$\sum_{i \in I} x_i = 1$$

(3) $$x_i \geq 0 \quad i \in I$$

where

J = criterion value

β = subjective weight on the variance of the return on the portfolio

x_i = the fraction of the portfolio invested in equity i

I = the set of equities

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \end{bmatrix} = \begin{bmatrix} 8 \\ 12 \\ 15 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{bmatrix} = \begin{bmatrix} 6 & -5 & 4 \\ -5 & 17 & -11 \\ 4 & -11 & 24 \end{bmatrix}$$

The name of the main program we will present to solve the Markowitz problem is `gaportfol1.m`, i.e. it is a genetic algorithm portfolio problem. This program and all the functions it calls are available in the book web page.

The basic structure of the program, shown below, is analogous to the program `gagame.m` presented earlier in the Genetic Algorithms and Evolutionary Games chapter. The program consists of three main parts. The first part contains the initialization of counters and parameters and a function call to initialize the population. The second part is a `for` loop across generations that in turn contains several function calls. Finally, the third part contains commands to print and graph the main results.

```
% initialization of counters and parameters;
nruns = 100;   popsize = 8;

beta = 2;
mu = [8 12 15]';
sigma = [6 -5  4;
        -5 17 -11;
         4 -11 24];

num = 3;   clen = num * 8;   pmut = 0.5;

% generation of chromosome strings of initial population
genepool= initpopdet(popsize);

for k = 1:nruns;

      % transformation of chromosome string into normalized n-asset
      % portfolio
      pwm = normport(genepool,popsize,clen,num);

      % computation of fitness function and fittest individual
      [fit, bestind, bestfit] =
         fitness_gaportfol(pwm,mu,popsize,beta,sigma);
      wbest(:,k) = bestind;
      fbest(k) = bestfit;

      % selection of parents;
      [parent0,parent1] = parentsdet(fit,genepool);

      % crossover of parents chromosome strings
      [child0,child1] = crossover(clen,parent0,parent1);

      % mutation of children chromosome strings
      for h = 1:2:popsize;
          child0mut = mutation(pmut,clen,child0);
          genenew(h) = child0mut;
          child1mut = mutation(pmut,clen,child1);
          genenew(h+1) = child1mut;
      end
    genepool= genenew;
end
```

```
% print and graph optimal weights and criterion;
wbest
fbest
figure(1);
xaxis = [1:1:nruns]';
plot(xaxis,wbest(:,:));
figure(2);
xaxis = [1:1:nruns]';
plot(xaxis,fbest(:,:));
```

In the initialization of counters and parameters section we set the number of runs `nruns`, the population size `popsize`, the parameters of the portfolio model (the risk aversion coefficient `beta`, the vector of mean returns `mu` and the variance-covariance matrix `sigma`) and the number of assets `num`. We also set the length of the chromosome string `clen` to be used to represent each portfolio as equal to the number of assets `num` times eight, that is, 24 bits. Finally, we set the probability of a child mutation `pmut`.

We then call the function `inipopdet` to initialize the vector `genepool` which will contain a number of portfolios equal to the population size, each portfolio represented by a 24-bit chromosome string. So, in our example, `genepool` is a vector of 24 bit chromosomes with an element for each of the eight individuals in the population.

Then we move on to the main `for` loop in the program, running from 1 to the number of runs. This loop contains a sequence of function calls. It starts with a call to the function `normport` to transform each 24-bit chromosome string corresponding to a portfolio into an equivalent normalized weight three-asset portfolio. The vector `genepool` may be thought of either as (1 x popsize) vector of integers or as a (1 x popsize) vector of 24 bit strings. Thus the `normport` function transforms the (1 x `popsize`) vector `genepool` into the (3 x `popsize`) portfolio weight matrix `pwm` which will contain, in each column, the normalized weights of each three-asset portfolio.

Next follows a call to the `fitness_gaportfol` function to compute the fitness function for each individual (each portfolio) and to select the fittest individual, which at each run will be stored in the `k`th column of the matrix `wbest` while the corresponding criterion value will be stored in the `k`th element of the vector `fbest`. Thus, at the end of the runs, these matrix and vector will contain the sequence of optimal portfolios and optimal criterion values respectively.

Next the function `parentsdet`, using the fitness function previously

computed, will select two parents (`parent0` and `parent1`) who will form a couple. This is followed by a call to the function `crossover` which will generate two children (`child0` and `child1`) as the product of the crossover of the chromosome strings of the two parents.

Next comes a `for` loop whose index goes from 1 to `popsize` in increments of two. In this loop, out of the two newborn children a new generation will be created through mutations of their chromosomes. Half of the new generation will come out of mutations of the first child (`child0`) and the other half will come out of mutations of the second child (`child1`). At every pass through the loop the function mutation is called which generates a mutated child, and its 24-bit chromosome representation is stored in a cell of the `genenew` vector. Once the new generation is created, the new vector `genenew` replaces the old vector `genepool` and the main loop of the program starts over again.

Finally, once the main loop goes through the established number of runs, the matrix of fittest individuals `wbest` and the vector of optimal criterion values `fbest` are printed and plotted.

This provides an overview of the program. It is important to point out that every time you run the program, particularly when changing the number of generations or the population size, you should clean out the old commands and workspace to avoid displaying spurious results. To do so, go to Edit in the top MATLAB menu. Then select Clear Command Window and confirm with Yes that you want to do this. Then do the same for Clear Command History and for Clear Workspace.

We will now present each function in detail.

4. Functions

4.1 Initpopdet

This function is simple in that all it does is to assigns the same portfolio weights (33%) to each asset in each portfolio as in the experiments performed in the Portfolio Model chapter so that we can easily compare results. However, the function is made more complicated by the necessity to represent a row vector of the three weights, i.e.

[33 33 33]

with a single 24 bit string such that each 8 bit segment of the string represents the number 33, i.e. 00100001. Thus the 24 bit string is

00100001 00100001 00100001

This string has ones in the positions 0, 5, 8, 13, 16 and 21 (counting from the right to left beginning with zero). Therefore the integer value of this string is

$$(1)\ 2^{21}+(1)\ 2^{16}+(1)\ 2^{13}+(1)\ 2^{8}+(1)\ 2^{5}+(1)\ 2^{0}$$

```
w = 2^21+2^16+2^13+2^8+2^5+2^0;
```

Thus the MATLAB for this initialization function is

```
function genepool = initpopdet(popsize);
w = 2^21+2^16+2^13+2^8+2^5+2^0;
dec2bin(w,24)
genepool = w * ones(1,popsize);
```

The header statement for the function, i.e.

```
function genepool = initpopdet(popsize);
```

tells us that the name of the function is `initpopdet`, that the argument `popsize` will be passed to the function and the result `genepool` will be returned by the function.

After `w` is defined in the first statement in the function, the statement

```
dec2bin(w,24)
```

appears. It does not play an essential role in the function since it only serves to print a binary representation of `w` for debugging purposes. Since there is no semicolon at the end, this statement will return and print the 24-bit binary representation of `w`. Finally, a (1 x `popsize`) vector of ones will be multiplied by the previously created cell `w` to obtain the initial population vector `genepool`. So

in the case at hand `genepool` is a vector of eight 24 bit strings. Finally, notice that an `end` statement is not necessary at the end of the function.

We will see later, in the `initpoprand_gaportfol` function, how to replace this rudimentary function by a more sophisticated random procedure to initialize the population.

4.2 Normport

This function takes the 24-bit chromosome string corresponding to each portfolio and creates an equivalent normalized three-asset portfolio representation. Thus, it transforms the (1 x `popsize`) vector `genepool` into the (3 x `popsize`) portfolio weight matrix `pwm` which will contain, in each column, the normalized weights (that is, the weights will add to one) of each three-asset portfolio. The complete function is listed below.

```
function pwm = normport(genepool,popsize,clen,num);
genetemp = genepool;
for i = 1:popsize;
      n = ceil(clen/num);
      mask = 2^n-1;    % note that this is 2^n minus one
      port = zeros(1,num);
      for j = 1:num
        port(num-j+1) = bitand(genetemp(i),mask);
        genetemp(i)= bitshift(genetemp(i),-n);
      end
      port = port/sum(port);
      pwm(:,i) = port';
end
```

Since we are using a chromosome with 24 bits and have 3 assets in the portfolio, we have eight bits to specify the proportion of the portfolio held in each of the assets. Since eight bits permits us to specify integers from 0 to 255 we will have an accuracy of about half a percent in the solution to the portfolio problem. So the first step after creating the temporary variable `genetemp` is to determine the number of bits, `n`, used for each equity by dividing the chromosome length by the number of equities. This is done with the statement

```
      n = ceil(clen/num);
```

where `ceil` is a MATLAB function that returns an integer that is the ceiling, i.e. the round off of a decimal to the nearest integer greater than the number. Also recall that clen is the chromosome length, 24, and num is the number of assets in the portfolio, 3, so in our case this becomes `ceil(24/3)` or eight. This statement assures that there will be an integer number of bits to represent the percentage of each stock held in the portfolio.

The next step is to pull out the `n` bits in the chromosome that correspond to the proportion for each equity. This is done by creating a mask in which the lower order `n` elements are ones and all other bits are zero. This is done with the statement

```
mask = 2^n-1;
```

Keep in mind the precedence rules so this is 2^n minus one. Thus in our case with `n` equal to 8, `mask` is an integer variable with value 255 and its binary representation is the 24-bit string

00000000 00000000 11111111

So we can use a `bitand` operation with this mask to pick off eight bit sections of the chromosome.

The next step is to initialize the vector which carries the percentage allocation of elements in the portfolio. In our case this is a three element vector of integers that is initialized with the statement

```
port = zeros(1,num);
```

This vector is then used in a `for j` loop over the number of equities in the portfolio to get the bit string for each element in the portfolio as follows

```
for j=1:num
    port(num-j+1) = bitand(genetemp(i),mask);
    genetemp(i) = bitshift(genetemp(i),-n);
end
```

The mask variable is used on each pass through the `for` loop to put the lower

order eight bit section of the chromosome into the variable `port`. Also in each pass the chromosome is shifted to the right by eight bits and filled with zeroes in the left most eight bits using the `bitshift` operation.

Thus if we begin with a chromosome like the following

00000110 11000000 11100111

The first pass through the loop would put the bit string

00000000 00000000 11100111

into the variable `port(3)` and the second pass would put the bit string

00000000 00000000 11000000

into the variable `port(2)`, etc.

Since the `port` variables are now integers with values between zero and 255 they must be normalized by the sum of their values to convert them to percentages of the portfolio. This is done with statement

```
port = port/sum(port);
```

Finally, the transposed of the three-element vector `port` is stored in the corresponding column of the matrix `pwm` with the statement

```
pwm(:,i) = port';
```

4.3 Fitness_gaportfol

This fitness_gaportfol function uses a procedure similar to the one used in the Portfolio Model chapter earlier in the book.

```
function [fit,bestind,bestfit] =
fitness_gaportfol(pwm,mu,popsize,beta,sigma);
pret = pwm' * mu;
for j = 1:popsize;
    pvar(j) = 0.5 * beta * pwm(:,j)' * sigma * pwm(:,j);
end
fit = pret - pvar';
[top topi] = max(fit);
bestind = pwm(:,topi);
bestfit = top;
```

It begins with a statement to compute the vector of returns `pret` as the product of the portfolios times the corresponding returns. Then statements are included to compute the vector of portfolio variance costs `pvar` and the corresponding criterion function vector, or fitness vector `fit`. So in our case the vector `fit` is an 8 element vector that provides the fitness level for each of the individuals in the population.

The statement

```
[top topi] = max(fit);
```

then returns the value (`top`) and the index (`topi`) corresponding to the maximum value in the fitness vector `fit`. Finally, the `topi` index is used to assign to the "best individual" vector, `bestind`, the corresponding normalized three-asset vector with the statement

```
bestind = pwm(:,topi);
```

while the corresponding value of the criterion function vector is assigned to the variable `bestfit` with the statement

```
bestfit = top;
```

With the fitness now determined we turn next to the selection of parents, crossover and mutation.

4.5 Parentsdet, Crossover and Mutation

These functions are exactly the same as the ones used in the Genetic Algorithms and Evolutionary Games chapter. They even have the same variable names. Thus they can be used by both the program in that chapter and the programs in this chapter.

5. Results

Figure 12.1 below shows the results of running the main program `gaportfol.m` with a number of runs equal to 100 and a population size of 8. The optimal values of the portfolio weights for the last run are w1 = 0.26, w2 = 0.42 and w3 = 0.32, which is slightly different than the results we obtained in the earlier chapter on portfolio models.

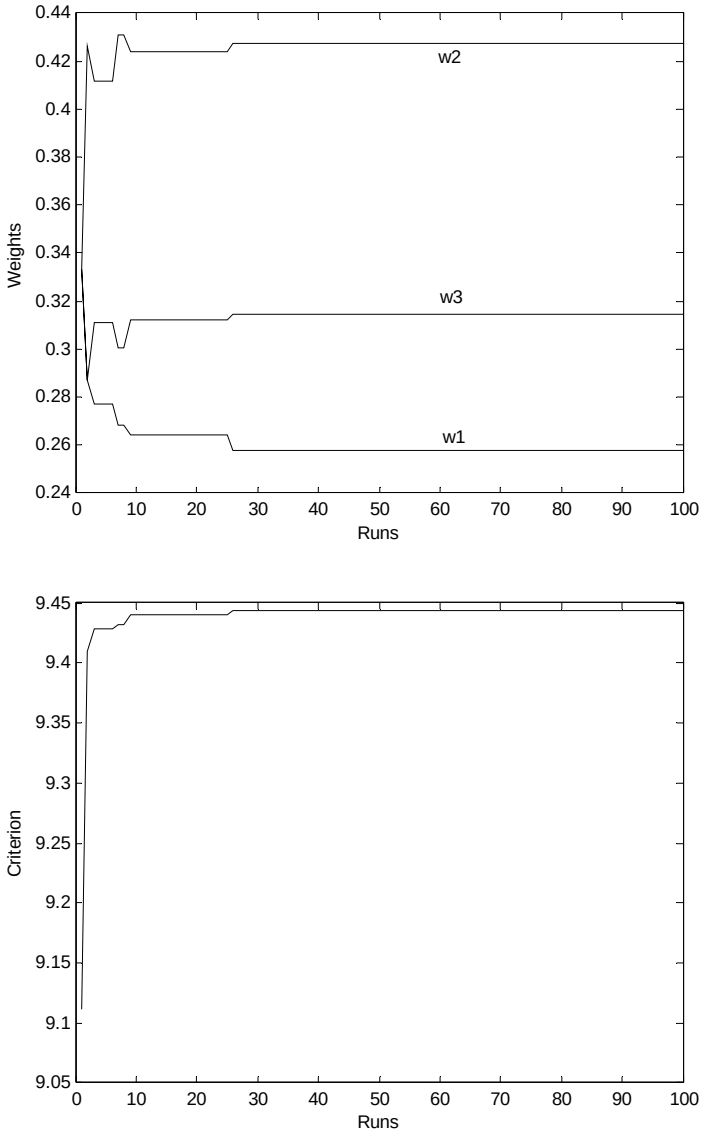


Figure 12.1 Genetic Algorithm Portfolio Example

We can see how, after starting from initial values equal to 0.33 the weights converge to the optimal values. We can see also how the criterion value converges to a value of 9.44.

6. Refinements

The program `gaportfol1.m` and its functions give us a basic idea of the work with genetic algorithms. However, some of its functions are quite rudimentary and they do not take us beyond what we learnt in the chapter on Genetic Algorithms and Evolutionary Games. In this section we will introduce some alternative and more sophisticated functions and main program structure. They can also be used to develop a more sophisticated version of the `gagame.m` program presented in the Genetic Algorithms and Evolutionary Games chapter.

6.1 Initpoprand_gaportfol

With the `initpopdet` function we generated, in a deterministic way, an initial vector of portfolios all with the same weights. However, it is customary in the field of genetic algorithms to generate the initial population randomly. To do so, we will introduce the `initpoprand_gaportfol` function shown below. In the chapter on Genetic Algorithms and Evolutionary Games we also generated the initial population randomly - in the `initpoprand_gagame` function - but with a different procedure. Here we present an alternative.

```
function genepool = initpoprand_gaportfol(clen,popsize);
mask = 1;
genepool = zeros(1,popsize);
for j = 1:popsize
    for i = 1:clen
        if (rand < 0.5)
            genepool(j)= bitor(genepool(j),mask);
        end
        genepool(j) = bitshift(genepool(j),1);
    end
    genepool(j) = bitshift(genepool(j),-1);
end
```

The key elements of this function are a set of two nested `for` loops and an `if` statement. The index of the `for` loop over individuals in the population is `j` and the index over the bits in the binary representation of the chromosome for each individual is `i`. Also the `i` loop runs from 1 to `clen` which is the number of binary elements in the chromosome. In the present case this is 24.

Also each bit in the chromosome for individual `j` will be modified, i.e. switched from zero to one with probability one half since `rand` is a zero-one uniform distribution random number generator.

Next focus for the moment only on the portion inside the `for` loop for the number of bits. The key element in this segment of code is the `bitor` operation. We know from the above that `genepool(j)` has been initialized to zero and the variable `mask` has been initialized to one, therefore the `bitor` operation applied to these two strings will yield

00000000 00000000 00000001

on the first pass through the `i` loop if the call to `rand` yields a value that is less than 0.5. Also we know that this will occur half the time.

Then following the `if` statement the operation `bitshift` is used to shift the binary string `genepool` one position to the left and to put a zero in the right most (lowest order) position. Thus after the `bitshift` operation the string above becomes

0000000 00000000 00000010

Thus in the `i` loop the bits in the `genepool(j)` string are considered one by one and changed from 0 to 1 with probability 0.5. Finally, one can see in the above code that `bitshift` is used to move the bit string one step back to the right after the end of the `i` loop. Otherwise the last bit in the string will always be equal to zero.

To use this function, we have to substitute in the program `gaportfol.m` the following statement for the call to the function `initpopdet` in the "generation of chromosome string of initial population" section

```
genepool = initpoprand_gaportfol(clen,popsize);
```

Figure 12.2 shows the results of running program `gaportfol2.m`, which is program `gaportfol1.m` with this function.

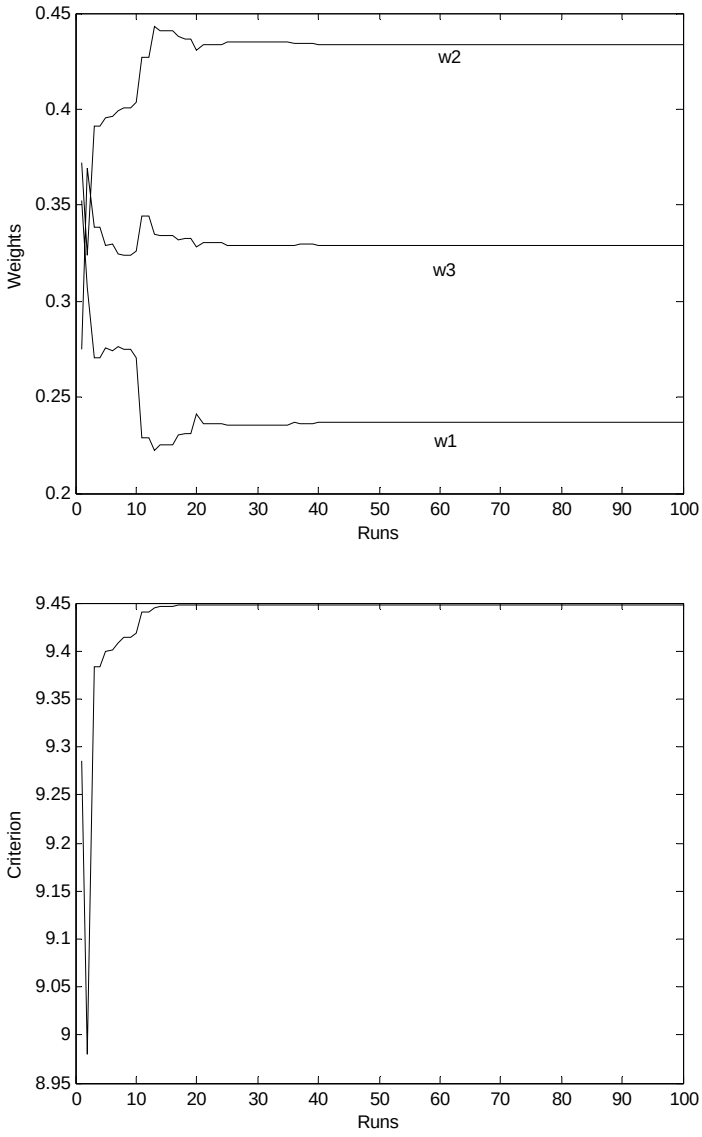


Figure 12.2 Genetic Algorithm with `Initpoprand_gaportfol` Function

The optimal portfolio weights are here w1 = 0.24, w2 = 0.43 and w3 = 0.33, which is slightly different than the results obtained in the earlier chapter on portfolio models. Also, as expected, the path of the optimal weights starts from random locations instead of starting from the 0.33 value as in Figure 12.1. The criterion value, after decreasing during the initial runs, converges to a value of 9.45.

6.2 Parentsrand

In programs `gaportfol1.m` and `gaportfol2.m` we used the function `parentsdet` to select the two parents of a new generation. That was a deterministic procedure where we picked as parents the two individuals with the highest and next to the highest value of the fitness function. However, the usual practice in the genetic algorithms field is to introduce some randomness in the selection of parents. The function `parentsrand` (parents random) we will present in this section is a first step in that direction.

The method to be used can be thought of as a cross between a pie chart and a roulette wheel. Consider a case in which there are five individuals in the population and all five have the same fitness level of 40. We could then use a pie chart to represent the percentage of the total fitness of the population of 200 which is held by each individual as shown in Figure 12.3.

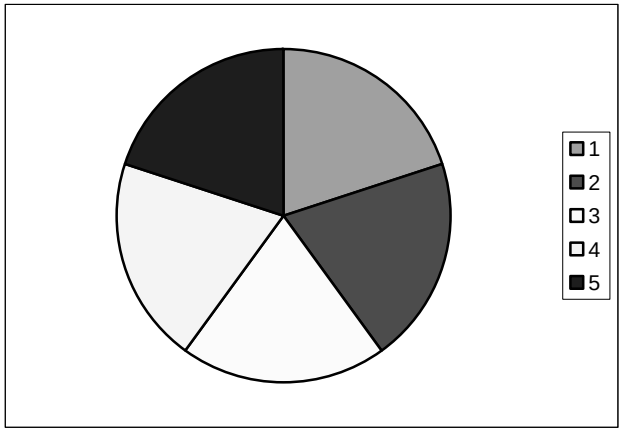


Figure 12.3 A Balanced Pie Chart / Roulette Wheel

One could also think of this pie chart as a roulette wheel which is spun each time a mate is to be selected. Since all the slices of the pie are the same size the probability of each individual being selected as a mate would be the same.

However, consider instead a case in which the fitness of each of the five individuals is different, say 20, 60, 30, 20, and 70. Then the pie chart representing the percentage of the total fitness held by each of the individuals would look like Figure 12.4.

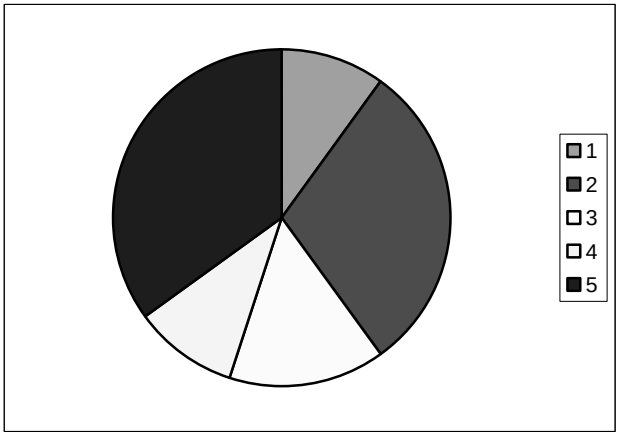


Figure 12.4 An Unbalanced Pie Chart / Roulette Wheel

In this case each time the roulette wheel is spun there would be decidedly different probabilities that each individual would be chosen.

The function code is shown below.

```
function f = parentsrand(fit,popsize,genepool);
cumfit = sum(fit);
val = 0;
spin_val = rand * cumfit;
j = 1;
while ((val < spin_val) & (j < popsize))
    val = val + fit(j);
    j = j + 1;
end
f = genepool(j);
```

The variable `cumfit` is the cumulative value of the individual fitness variables, i.e. the sum of the fitness levels of the members of the population. The variable `val` is used to move around the roulette wheel as it is spun and the variable `spin_val` carries the information about how far the roulette wheel travels before it stops. The variable `rand` provides a number from the zero to one interval of a uniform distribution Therefore each time the roulette function is called `spin_val` takes on a different value that ranges uniformly from zero to the sum of the fitness levels. However, the slices of the roulette wheel are not all the same size; rather they represent the relative fitness of the individuals. So more fit individuals are more likely to be chosen.

The selection occurs in the `while` loop which repeats until the variable `val` exceeds `spin_val` or until the loop index `j` exceeds the population size. Also, each time through the loop the variable `val` is augmented by the fitness level of individual `j`. After the completion of the `while` loop the index of the selected individual is then used to select the corresponding chromosome string from the vector `gen` and this information is transferred to the variable `f` which will be the output of the function.

One thing to notice about this procedure is that it is "sampling with replacement", i.e. when a mate is chosen that individual is not removed from the population but rather is left in the population and is eligible to be chosen on subsequent calls to the function.

To use this function, we have to replace in the program `gaportfol.m` the call to the function `parentsdet` with the call to the function `parentsrand` in the "selection of parents" section in the following two statements:

```
parent0 = parentsrand(fit,popsize,genepool);
parent1 = parentsrand(fit,popsize,genepool);
```

Figure 12.5 shows the results of running program `gaportfol3.m`, which is program `gaportfol1.m` with this function and also using the `initpoprand_gaportfol` function as explained in the previous section.

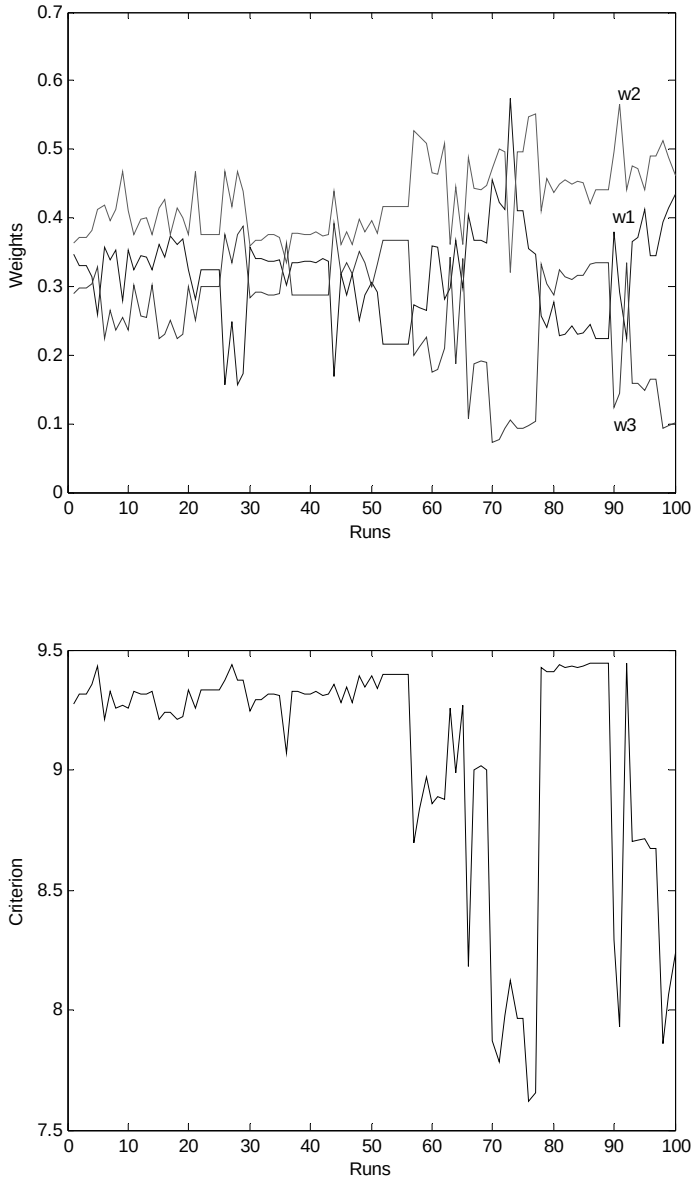


Figure 12.5 Genetic Algorithm with `parentsrand` and `initporand_gaportfol`

We see here that the patterns of optimal portfolio weights and criterion values, at variance with Figures 1 and 2, are very unstable. Why? The reason may be that here we are choosing the two parents with some randomness, while in the other two cases we always chose the two best performing individuals and with them formed a couple. This result is important to make the point that genetic algorithms are designed to perform a wide exploration of the solution space. Thus, they usually work with large populations

and a large number of runs. If we increase those values we will observe some performance improvement, i.e. with 100 runs as before but now with a population of 500 instead of 8. However, before doing so it will be convenient to adjust the program `gaportfol3.m` to widen the pool of best couples as we will see in the next section.

6.3 Selecting More Than One "Best" Couple

In the program `gaportfol3.m` we introduced randomness in the initialization of the population and in the selection of parents. However, we kept using the same procedure we used in the previous programs for the process of parents selection, that is, we just selected the couple with the highest criterion value to give birth to the entire new generation. However, to make a better use of the random selection of parents process introduced in the previous section, that is, to widen the search space of an optimum thus reducing the changes of being trapped in a local optimum, it may be convenient to obtain the new generation of children from more than just one couple. This can be accomplished, for example, by extending the range of the children's generation loop in the "mutation of children chromosome strings" section from program `gaportfol3.m`

```
% mutation of children chromosome strings
    for h = 1:2:popsize;
    child0mut = mutation(pmut,clen,child0);
    genenew(h) = child0mut;
    child1mut = mutation(pmut,clen,child1);
    genenew(h+1) = child1mut;
end
```

to make it cover the two previous sections also, namely to include also the calls to the `parentsrand` and `crossover` functions.

```
for h = 1:2:popsize;
      % selection of parents;
      parent0 = parentsrand(fit,popsize,genepool);
      parent1 = parentsrand(fit,popsize,genepool);

      % crossover of parents' chromosome strings
      [child0,child1] = crossover(clen,parent0,parent1);

      % mutation of children chromosome strings
      child0mut = mutation(pmut,clen,child0);
      genenew(h) = child0mut;
      child1mut = mutation(pmut,clen,child1);
      genenew(h+1) = child1mut;
end
```

With this change, at each pass of the loop, each "best" couple randomly selected in the "selection of parents" section will give birth to only two children who in time will experience mutations. Figure 12.6 below shows the result of an experiment with program `gaportfol4.m`, which incorporates this change, and for 100 runs with a population of 500.

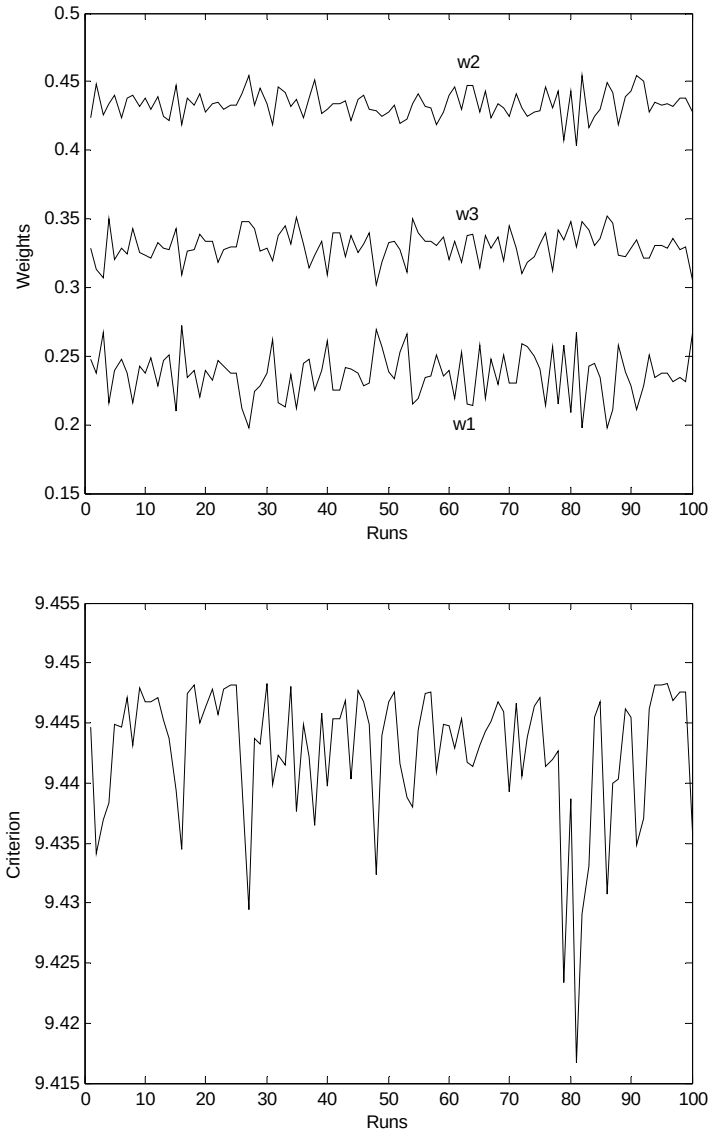


Figure 12.6 Example with Many "Best" Couples

We see that the performance improves in the sense that the weights and the criterion values follow a more discernible pattern, but it is still more unstable than in Figures 1 and 2. However, as we said above, the risk of being trapped in a local optimum in the case a number of them exists is expected to be lower.

7. A More Difficult Portfolio Problem

So far we have been working with a Markowitz type portfolio optimization problem with a quadratic criterion function. This is by nature a convex problem with a single optimum. However, genetic algorithms are usually employed to solve more complex problems, that may have a number of local optima and be difficult for local optimization methods such as gradient or Newton methods to solve, but easier for global optimization methods such as genetic algorithms.[30]

In the previous convex problem the criterion (fitness) value was the mean return (revenue) minus the variance cost, ignoring the brokerage cost for purchasing equities. In this problem we change that to the *profit* minus the variance cost where the profit is the mean return less the brokerage fee. Moreover we use a realistic form of the brokerage fee that includes both a fixed and a marginal cost for the purchase of each type of equity. This has the effect of making the average cost of purchasing equities decline with the number of equities purchases and this in turn raises the possibility that the optimization problem may have local optima. Also we have imposed the restriction that the individual must purchase a percentage above some lower bound of each type of equity. Thus if the selection of parents, crossovers and mutations generate a portfolio in which one or more equities are below the lower bound this amount is reset to zero and the small amount is redistributed to the other stocks.

The code itself for this second problem is similar to that for the problem in the previous section with the exception of the `fitness_gaportfol` function which will now be replaced with the `fitnessnc` function. This function has two main parts. The first one, dealing with the portfolio redistribution, is shown below.

[30] For some other approaches to global optimization see Goffe (1996) and Tucci (2002).

```
function [fit,bestind,bestfit] =
fitnessnc(pwm,mu,popsize,beta,sigma,num);

% Portfolio Redistribution
for j = 1:popsize;

    cond = ones(num,popsize);
    counter = num;
    acum = 0;

    for i = 1:num;
        if pwm(i,j) < 0.1
            acum = acum + pwm(i,j);
            pwm(i,j) = 0;
            cond(i,j) = 0;
            counter = counter - 1;
        end
    end

    for i = 1:num;
        if counter > 0
            if cond(i,j) == 1;
                pwm(i,j) = pwm(i,j) + (acum / counter);
            end
        end
    end

end
```

There is a main `for` loop, running across portfolios from 1 to `popsize` - the population size - and two inside loops running across each portfolio from 1 to `num` - the number of equities. The main loop begins by defining three auxiliary variables that will be reset at each pass of the loop through each portfolio. The variable `cond` is defined as comprised of ones and with the same dimensions as the variable `pwm`. It is initialized with ones and will be used to mark with ones those equities whose amount is above the allowed lower bound and with zeroes otherwise. The variable `counter` will be used to count the number of equities in each portfolio whose amount is above the lower bound. The variable `acum` will contain the accumulated amount of stocks below the lower bound in each portfolio.

The first inside loop follows. When the amount of an equity in the portfolio is below the lower bound - set to 0.1 in the present example - that amount is accumulated to be later redistributed. Then, that equity's participation in the portfolio is set to zero and marked with a zero in its corresponding location in the `cond` matrix. Finally, the counter of the number of equities above the lower bound in the portfolio is decreased in one unit.

The second inside loop begins by checking that the variable `counter` is different from zero to later avoid a possible division by zero. Then a conditional statement checks if the corresponding equity is marked with a one, thus being above the lower bound. In this case, the corresponding proportional amount of previously accumulated stocks to be redistributed is added to that equity.

The second main part of the function, dealing with the computation of portfolio returns and the selection of the best portfolio is shown below. It is very similar to the `fitness_gaportfol` function corresponding to the convex example, with some minor differences.

```
% Computation of portfolio returns and best portfolio
fc = 0.2 * ones(1,num);
mc = 0.05 * ones(1,num);
pret = pwm' * mu;
for j = 1:popsize;
    pvar(j) = 0.5 * beta * pwm(:,j)' * sigma * pwm(:,j);
end
for j = 1:popsize;
    pbrok(j) = fc * cond(:,j) + mc * pwm(:,j);
end
fit = pret - pvar' - pbrok';
[top topi] = max(fit);
bestind = pwm(:,topi);
bestfit = top;
```

The first difference is that it is necessary to define vectors of fixed and marginal cost terms for the brokerages fees. This is done in the initialization section of the function with the statements

```
fc = 0.2 * ones(1,num);
mc = 0.05 * ones(1,num);
```

Thus in these vectors we allow for different fixed costs and marginal costs for the various types of equities. However, we have treated these costs as the same for

across equities in the present example. We have purposely made the fixed cost relatively large in order to increase the likelihood that the problem will have local optima. Then we compute the brokerage cost for each portfolio with the loop

```
for j = 1:popsize;
    pbrok(j) = fc * cond(:,j) + mc * pwm(:,j);
end
```

where `pwm` is the matrix containing each vector of portfolio weights and where `cond` is the matrix containing each vector of portfolio marks, with ones for equities above the lower bound and zeros otherwise. Thus, the fixed cost will be charged on portfolios above the lower bound only. Finally, as in the `fitness_gaportfol` function, we compute the fitness of each individual (now including the brokerage cost) and select the fittest one.

To solve this nonconvex example we use a modified version of the program `gaportfol4.m`, which we will name `gaportfol5.m`. In the "computation of fitness function and fittest individual" section of the `gaportfol4.m`, we have to replace the previous `fitness_gaportfol` function call with the statement

```
[fit, bestind, bestfit] =
    fitnessnc(pwm,mu,popsize,beta,sigma,num);
```

Figure 12.7 shows the result of running the program `gaportfol5.m`.

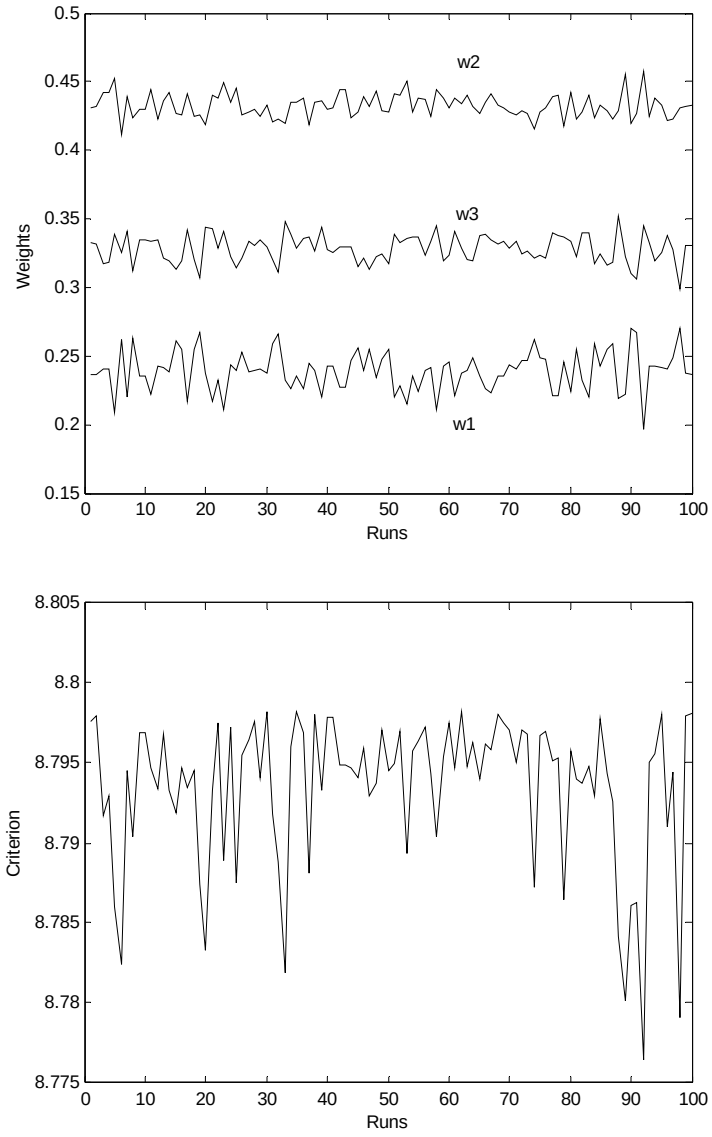


Figure 12.7 Nonconvex Problem

The results are similar to those shown in Figure 12.6. However, they may change significantly if we introduce substantial changes in the fixed costs and/or in the equities lower bound.

8. Experiments

Just as in the Portfolio Model chapter, the simplest experiments with this genetic algorithm code are to change the means and/or variances of the portfolio and/or the brokerage costs and see how the weights of the best portfolios change in response. Another simple experiment would be to change the number of model iterations and/or population size to see how this affects the outcome.

A more interesting set of experiments will be to introduce further refinements in the code to move closer to the actual practice in the field of genetic algorithms. A first refinement would be to introduce mutations in more than one bit in the children's chromosomes in the `mutation` function. A second refinement would be to introduce more than a single crossover point in the `crossover` function.

9. Further Reading

A classic reference in the genetic algorithm literature is Goldberg (1989). For financial applications, see Bauer (1994).

Chapter 13

Macroeconomics in GAMS[31]

Macroeconomic models study the behavior of economic systems from an aggregate point of view. They try to capture the interdependence between consumption and investment expenditure, fiscal and monetary policy variables, the price level, the aggregate supply and the level of employment. From a modeling point of view, we can say that there are three main classes of macroeconomic models: standard models, rational expectations models and intertemporal optimization models. Standard models like the one used in this chapter, which are also known as IS-LM models, specify aggregate relationships to explain the behavior of macroeconomic variables. Also, they usually assume that economic agents form expectations in an adaptive way. Rational expectations models also work with aggregate relationships, but they assume that the economic agents display forward looking behavior. That is, in order to form expectations, those agents are assumed to make use of all the available information, including the model of the economy that policymakers use to model their behavior. Finally, intertemporal optimization models share with rational expectations models the same assumptions in connection with expectation formation, but try to base their modeling of macroeconomic behavior on more explicit "microfoundations".

IS-LM models are the backbone of almost all introductory and intermediate macroeconomics textbooks and have been for a long time the main workhorse in the field of empirical macroeconomics, as is the case, for example, of the Fair model

---

[31] This chapter draws extensively on both the verbal and the mathematical development in Mercado, Kendrick and Amman (1998). Kluwer Academic Publishers have kindly granted us permission to reuse here substantial materials from our previously published paper.

(http://fairmodel.econ.yale.edu/).[32] An example of a well-known rational expectations model is the Taylor (1993) model. Finally, intertemporal optimization models are still relatively small and are not used very much in large scale empirical applications or policy analysis. They are mainly used for teaching at the graduate level, for experimental purposes or for policy analysis exercises at a relatively small scale. One of the most influential models of this type is the one by Rotemberg and Woodford (1997).

The solution methods of the models mentioned above critically depend on the assumption regarding expectations formation. For example, models with backward looking expectations, like those in the standard-type IS-LM model to be presented in this chapter, are solved using a given set of initial conditions for the lagged variables and paths for policy and exogenous variables. As we will see later in the book, this is not the case for rational expectations and intertemporal optimization models, since they share the assumption of forward looking behavior and present what is technically known as "two-point boundary value problems". To be solved they require both initial and terminal conditions or specific iterative procedures.

1. The Hall and Taylor Model

In this chapter we will introduce the Hall and Taylor (1997) model, a well known textbook standard model, and we will illustrate how to represent and simulate this model in GAMS. This is a twelve-equation nonlinear dynamic model for an open economy with flexible exchange rates. It is well suited to teach simulation and policy analysis at the undergraduate level. The core of this model can be seen as a standard IS-LM-Open Economy sub-model for the aggregate demand of the economy together with an "expectations augmented" Phillips Curve, that is, the aggregate supply. The Hall and Taylor model contains the equations, variables and parameters listed below.

Equations

IS-LM

(1) GDP identity $Y = C + I + G + X$

(2) Disposable Income $Y^d = (1-t)Y$

(3) Consumption $C = a + bY^d$

(4) Investment $I = e - dR$

[32] The antecedents of these models go back to the work of Keynes (1936) and Hicks (1937).

(5) Money Demand $M/P = kY - hR$

Expectations Augmented Phillips Curve

(6) Expected Inflation $\pi^e = \alpha\pi_{-1} + \beta\pi_{-2}$

(7) Inflation Rate $\pi = \pi^e + f\{(Y_{-1} - Y_N)/Y_N\}$

(8) Price Level $P = P_{-1}(1+\pi)$

Foreign Account

(9) Real Exchange Rate $EP/P_W = q + vR$

(10) Net Exports $X = g - mY - nEP/P_W$

Government Deficit and Unemployment

(11) Government Deficit $G_d = G - tY$

(12) Unemployment Rate $U = U_N - \mu\{(Y - Y_N)/Y_N\}$

Endogenous Variables

C : Consumption

E : Nominal Exchange Rate (foreign currency / domestic currency)

G_d: Government Deficit

I : Investment

P : Domestic Price Level

R : Real Interest Rate

U : Unemployment Rate

X : Net Exports

Y : GDP

Y^d: Disposable Income

π : Inflation Rate

π^e : Expected Inflation

Policy Variables

G : Government Expenditure

M : Money Stock

Exogenous Variables

P_w : Foreign Price Level

U_N: "Natural" Rate of Unemployment

Y_N: Potential GDP

Parameters

$a = 220;\ b = 0.7754;\ d = 2000;\ e = 1000;\ f = 0.8;\ g = 600;\ h = 1000;\ k = 0.1583;$
$m = 0.1;\ n = 100;\ q = 0.75;\ t = 0.1875;\ v = 5;\ \alpha = 0.4;\ \beta = 0.2;\ \mu = 0.33;$

The model is dynamic - all variables without subscripts correspond to time "t", those with "-1" subscripts correspond to "t-1", and so on. Also the model is nonlinear - nonlinearities appear in equation (5), (8), (9) and (10). As we will see later, its dynamic behavior displays the "natural rate" property: nominal shocks may affect real variables in the short-run, but not in the long run.

Eqs. (1) to (5) are standard in most macroeconomics textbooks. Eq. (1) is an identity that states that GDP always equals the sum of its main components: consumption, investment, government spending and net exports (exports minus imports). Eq. (2) determines disposable income as equal to GDP net of taxes. Eq. (3) is a standard consumption function in which current consumption depends on current income. Eq. (4) determines investment as an inverse function of the real interest rate. Finally, Eq. (5) defines real money balances as a positive function of income (money demand for transaction purposes) and a negative function of the interest rate (the opportunity cost of holding money instead of interest bearing assets).

Eqs. (6) to (8) correspond to an expectations augmented Phillips curve. Eq. (6) gives the expected inflation as a function of the past inflation in the last two periods (years). Eq. (7) determines the inflation rate as a positive function of the expected inflation rate and the GDP gap (the difference between actual GDP in the previous year and potential GDP). A positive gap means an overheated economy thus inflationary pressure. A negative gap means recession thus deflationary pressure. Eq (8) just defines the price level as a function of the price level the previous year and the inflation rate.

Eqs. (9) and (10) are foreign account equations. Notice that the nominal exchange rate *E* is defined as foreign currency / domestic currency. Thus an increase (decrease) in *E* is a nominal appreciation (depreciation) of the domestic currency. Eq. (9) determines the real exchange rate (the nominal exchange rage times the domestic price level divided the foreign price level) as a positive function of the interest rate. Thus, for example, an increase in the US interest rate (implicitly assuming that the interest rate in the rest of the world remains the same) will cause capital inflows and an appreciation of the dollar. Eq. (10) gives net exports as a function of GDP and the real exchange rate. Changes in GDP affect the demand for imports while exports do not change as much. Thus net exports will change. The real exchange rate is the relative price between domestic and foreign products. Thus its changes will affect imports and exports.

Finally, Eqs. (11) and (12) give the government deficit and the unemployment rate, and they have not feedbacks on the rest of the model.

It is usual to develop a compact graphical representation of a model like this in two graphs: and IS-LM graph and an aggregate demand-aggregate supply graph. To derive the IS schedule we substitute Eq. (2) into Eq. (3), Eq. (9) into Eq. (10), then Eqs. (3), (4) and (10) into Eq. (1). Solving the resulting equation for the interest rate we obtain

$$(13) \qquad R = \frac{a + e + g - nq}{d + nv} - \frac{1 - (b - t) + m}{d + nv} Y + \frac{1}{d + nv} G$$

This equation shows R as a function of Y (given G) and represents all the combinations of interest rate and income for which spending balances. To derive the LM schedule we just solve for R Eq. (5), obtaining

$$(14) \qquad R = \frac{k}{h} Y - \frac{1}{h} \frac{M}{P}$$

This equation also shows R as a function of Y (given M and P) and represents all combinations of interest rate and income for which the money market is in equilibrium. Finally, the graphical representation of both schedules in the (R,Y) space is shown in Figure 13.1. Given the model coefficient values, the IS curve will be downward slopping and the LM curve will be upward slopping. The intersection of the two schedules determines the equilibrium interest rate and income.

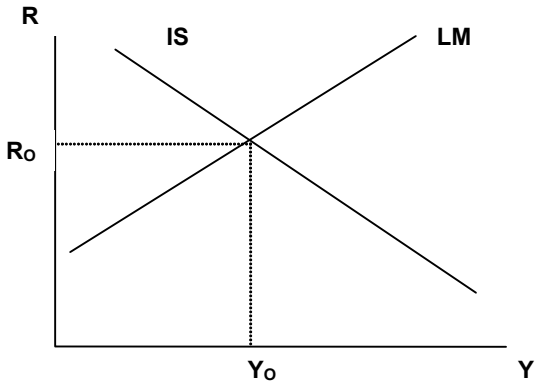


Figure 13.1 IS-LM Graph

The aggregate demand (AD) schedule represents the IS-LM part of the model in a different space: the price level (P) and income (Y) space. It shows how much people will demand at a given level or prices. It can be obtained combining Eqs. (13) and (14) and the result, given the values of the model coefficients, is a downward slopping nonlinear schedule with P as a function of Y as shown below in Figure 13.2. The aggregate supply is an Expectations Augmented Phillips Curve embodied in Eqs. (6), (7) and (8). To capture its behavior, we represent it in the (P,Y) space by means of two lines. The Y_N vertical line represents the long-run aggregate supply that is the potential or "natural" income level, which is assumed to be constant in the short-run. Finally, the horizontal line or "price line" (P) represents the short-run aggregate supply, which is supposed to be perfectly elastic, though in other textbook presentations it is assumed to be upward slopping. Figure 13.2 shows the graphical representation of aggregate demand and supply.

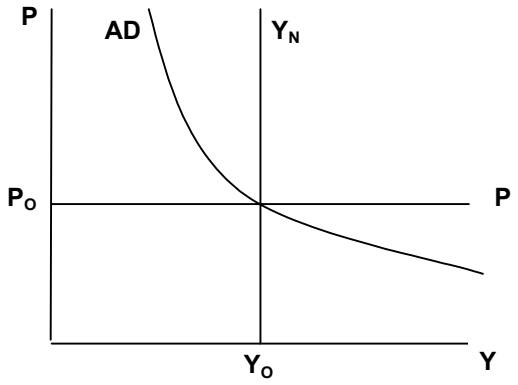


Figure 13.2 Aggregate Demand - Aggregate Supply Graph

The analysis of the effects of an increase in the money supply (M) will help us to understand the workings of the model in qualitative terms. An increase in the money supply will bring about disequilibrium in the money market, shifting the LM schedule to the right, thus bringing down R and increasing Y. This implies that the AD schedule also shifts to the right, as it is shown in Figure 13.3.

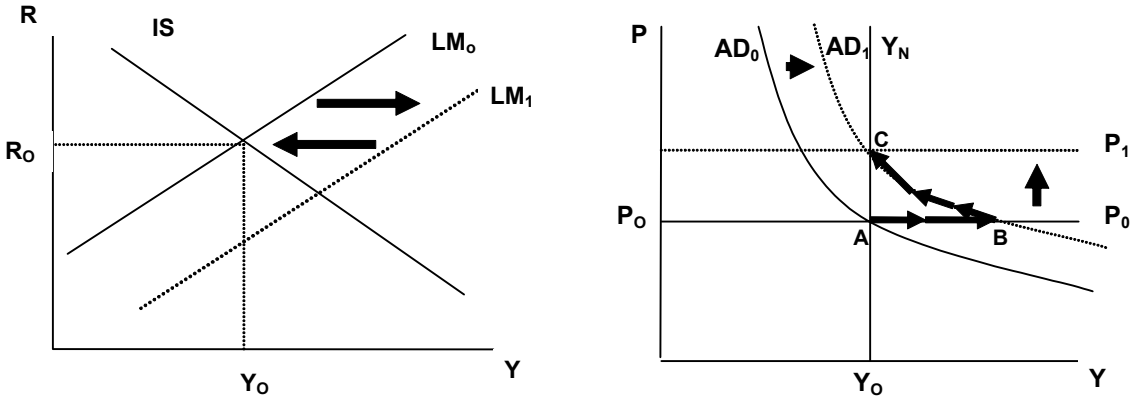


Figure 13.3 Qualitative Effects of an Increase in the Money Supply

In the short run prices are sticky, thus the economy moves from point A to point B. However, in the medium run, since there is a positive GDP gap, the inflation rate becomes positive and prices begin to increase, as can be seen in equation (7).

(7) Inflation Rate $\pi = \pi^e + f\left\{\left(Y_{-1} - Y_N\right)/Y_N\right\}$.

This process continues given that agents' expectations will change due to past changes in the inflation rate, as shown in equation (6).

(6) Expected Inflation $\pi^e = \alpha\pi_{-1} + \beta\pi_{-2}$.

As prices increase, real money balances decrease (see equation (5) below) shifting the LM schedule to the left.

(5) Money Demand $M/P = kY - hR$.

Finally, the economy moves from point B to point C. We can see then that the increase in the money supply was neutral in the long-run with respect to real variables, but not in the short-run.

2. The Hall and Taylor Model in GAMS

Different strategies can be followed when one is confronted with the problem of solving and performing policy experiments with a model like this. In the following, we will review some of them.

Usually, the first step in the analysis of a model like Hall and Taylor's is to find the steady-state values of the endogenous variables for a given set of constant values of the policy and exogenous variables. This requires the transformation of the model from dynamic to static. Solving a nonlinear system of equations, even when it is static, is not easy. In general, we have to rely on numerical techniques which may or may not deliver a solution, even if it exists, depending on the initial conditions provided. However, the model of our interest does not contain many or very strong nonlinearities, making the task of finding a solution relatively easy.

To solve for the steady-state, we have to eliminate all time subscripts and solve the resulting static nonlinear model. This does not present any challenge to GAMS users, even for beginners. Since this model is relatively straightforward we will not discuss it further here but rather turn our attention to the dynamic nonlinear model that is of greater interest. The file for this model is `htsim.gms` on the web site. It is also contained in Appendix 11A at the end of this chapter. We will discuss here in the body of the chapter two unusual aspects of the GAMS representations of this model. However, before doing so it is useful to look at the main `SET` specification of the model, namely

```
SETS  T    EXTENDED HORIZON     / 0*15 /
```

Thus the model includes sixteen time periods – zero, one, two through fifteen. Also, keep in mind that GAMS is not case specific and one will find the set of time periods specified in the GAMS statement at times as `T` and at other times as `t`; however they are the same.

Next we consider the way the dynamic variables and equations of the Hall and Taylor model are represented in GAMS. This is shown below. Note that to avoid notational conflicts in the GAMS statement, the mathematical parameters *e*, *g*, *m* and *t* have been renamed as `ee`, `gg`, `mm` and `tax`, respectively. Also, variables and parameters names denoted with Greek symbols in the mathematical statement of the model will be renamed in the GAMS statement, since GAMS does not handle such notation. Finally, the listing below does not include all the variable names or equations names that are in

the GAMS version of the model, but rather only a few. The list below does however, contain all the equations.

```
VARIABLES
Y(t)         gdp
Yd(t)        disposable income
   ...
EQUATIONS
eq1(t)     gdp identity
eq2(t)     disposable income
    ...;

eq1(t+2)..   Y(t+2)   =E=   C(t+2) + I(t+2) + G(t+2) + X(t+2) ;
eq2(t+2)..   Yd(t+2)  =E=    (1 - tax) * Y(t+2) ;
eq3(t+2)..   C(t+2)   =E=    a + b * Yd(t+2) ;
eq4(t+2)..   I(t+2)   =E=    ee - d * R(t+2) ;
eq5(t+2)..   M(t+2) / P(t+2)  =E=   k * Y(t+2) - h * R(t+2) ;
eq6(t+2)..   piex(t+2)=E=    alpha * pi(t+1) + beta * pi(t) ;
eq7(t+2)..   pi(t+2)  =E=    piex(t+2) + f*(Y(t+1)-Yn(t+2))/Yn(t+2) ;
eq8(t+2)..   P(t+2)   =E=    P(t+1) * (1 + pi(t+2)) ;
eq9(t+2)..   E(t+2) * P(t+2) / Pw(t+2)  =E=  q  + v * R(t+2) ;
eq10(t+2)..  X(t+2)   =E=   gg - mm*Y(t+2) - n*(E(t+2)*P(t+2)/Pw(t+2));
eq11(t+2)..  Gd(t+2)  =E=    G(t+2) - tax * Y(t+2) ;
eq12(t+2)..  U(t+2)   =E=    Un(t+2) - mu*(Y(t+2)-Yn(t+2))/Yn(t+2) ;
```

Notice that all variables and equations are defined over the set `t`. However, the model equations are specified over the set `t+2` and contain variables defined over the sets `t+2`, `t+1` and `t`, instead of following the corresponding original indices `t`, `t-1` and `t-2` respectively. This is due to the way in which GAMS handles the assignment of values to lagged variables.

For example, we could define the set `t` as:

```
SETS t /0,1,2,3/
```

and then write equation 6 with time subscripts as in its original formulation:

```
eq6(t)..   piex(t) =E=    alpha * pi(t-1) + beta * pi(t-2) ;
```

Then, when solving the model, GAMS would assign the default value zero to expressions like `pi(t-1)` and `pi(t-2)`, since `-1` and `-2` do not belong to the set `t`. Therefore, we

would not be able to assign to the inflation rate initial values other than zero, even if we wished to do so.

Thus, when dealing with models containing lagged variables in GAMS, we follow the following rule of thumb: for a solution horizon of duration t, specify equations starting from the longest lag. In Hall and Taylor's model, the longest lag is equal to 2. Notice how we wrote the model equations containing lags - eqs. 6, 7 and 8 - where we have variables with subscripts equal to `t`, `t+1` and `t+2`. At the same time, in equations containing no lags, all variables have subscripts equal to `(t+2)`. By operating in this way we "keep" the first two time periods (`t` and `t+1`) free to assign initial values and let GAMS find a solution for the remaining periods. More details on this are provided in Appendix D.

To complete the GAMS specification of Hall and Taylor's model, besides defining – as we did above – the extended horizon for simulations, we have to provide initial conditions for output and inflation.

```
SETS  t   EXTENDED HORIZON  / 0*15 /
      t0(t)      PERIOD ZERO
      t1(t)      PERIOD ONE;
      t0(t) = YES$(ORD(t) EQ 1);
      t1(t) = YES$(ORD(t) EQ 2);
```

With this specification, we are defining a fifteen-period time index as the set `t`. Then, we declare and define the subsets `t0` and `t1` and assign to them, respectively, the first and second elements of the `t` set - that is, the elements in the "ordinal 1" and "ordinal 2" places. Thus the GAMS statement

```
t0(t) = YES$(ORD(t) EQ 1);
```

can be read as "assign to the set `t0` the elements of the set `t` such that the ordinal position of element `t` is equal to one". The `$` operator in GAMS can be read as a "such that" operator in this context.

The specification for the sets `t0` and `t1` used above is useful in case one decides to change the extension of the simulation horizon, since we would not have to change the definition of the initial conditions subsets.

In the same way, we can also define terminal conditions subsets. These conditions become necessary in models containing rational expectations, as we will see later in the book. For instance, terminal conditions for the last and the previous-to-the-last period can be written by defining two new subsets - for example, `tf(t)` and `tf1(t)` - of the set `t` and then adding the following two expressions:

```
tf(t)   =  YES$(ORD(t) EQ CARD(t));
tf1(t) =  YES$(ORD(t) EQ (CARD(t) - 1));
```

where, as before, `ORD(t)` means "ordinal" and where `CARD(t)` means the cardinality, i.e. the number of elements in the set.

Next we turn our attention from the specification of the dynamics of the model in GAMS to the specification of the policy variable time paths. This is unusual in that the policy variables are specified in percent deviations from base levels rather than in levels. This is accomplished by providing statements which set the percent difference. An example is the statements that are used for monetary policy. They are

```
SETS
TS1(T) periods for shock 1    / 4*15 / ;
```

that creates a set `TS1` over which the policy change is defined and

```
Mper(TS1) = 0.0 ;
```

that sets the percentage change. Thus to create a solution where the money supply is 3 percent above the base level in periods `4` thru `15` one would modify the statement above to

```
Mper(TS1) = 0.03 ;
```

Alternatively, the user might want to have two periods in which the policies were above and then below that the base level. This would be done by first creating the two sets of time periods with GAMS statements of the form

```
SETS
TSPER1(T) Quarters in period 1    / 5*8 /
TSPER2(T) Quarters in period 2    / 10*13 / ;
```

Followed by statements to set the percent deviations, i.e.

```
Mper(TSPER1) = 0.03 ;
Mper(TSPER2) = -0.02 ;
```

Then the money supply would be 3 percent above the base level in quarters `5` thru `8` and 2 percent below the base level in quarters `10` thru `12`. However, when doing this be careful not to use quarters beyond those included in the set `T`.

The initial conditions for output and inflation are defined as:

```
Y.fx(t1) = ini1;   Pi.fx(t0) = ini2;   Pi.fx(t1) = ini3;
```

where `t0` and `t1` mean "period 0" and "period 1" respectively, "`.fx`" tells GAMS to keep the assigned values fixed during the execution of the program and `ini1` to `ini3` are given initial values.

In this model, in order to solve a system of equations in GAMS, it will be necessary to add an additional variable (`J`) and an additional equation (`JD`) and to maximize or minimize the added variable. Thus the `SOLVE` statement will be

```
SOLVE NONLDYN MINIMIZING J USING NLP;
```

Also, since the model contains indexed equations a stacking method is used in GAMS as discussed in Appendix H. Finally, since Hall and Taylor's is a nonlinear model, we have to invoke a nonlinear programming (NLP) solver. For an introduction to this type of solvers see Appendix F.

To perform simulations with this model we change the values of the policy variables or the parameter values, as discussed above, and compare the different dynamic solution paths obtained for the endogenous variables.

The graphical analysis we performed earlier gave us a useful representation of the qualitative behavior of the key variables of the economy. However, to deal with more variables and to obtain precise quantitative results, we have to simulate the model computationally. Figure 13.4 displays the results of two experiments: a first experiment where we start from an equilibrium position and then increase the money supply by 10% and a second experiment where we start from equilibrium and we increase government expenditure by 10%. Both increases are assumed to take place in period four and be permanent, that is, once they happen they are not reversed. Figure 13.4 shows the solution paths for income, the inflation rate, the interest rate and the nominal exchange

rate. The value of the variables between periods zero and three corresponds to the model steady state values. The continuous line corresponds to the money supply experiment, while the dotted line corresponds to the government expenditure experiment. GDP values are in billions of dollars. For the real interest rate and the inflation rate a value of 0.01 corresponds to 1%. The nominal exchange rate values correspond to an index value set equal to one in the steady state.

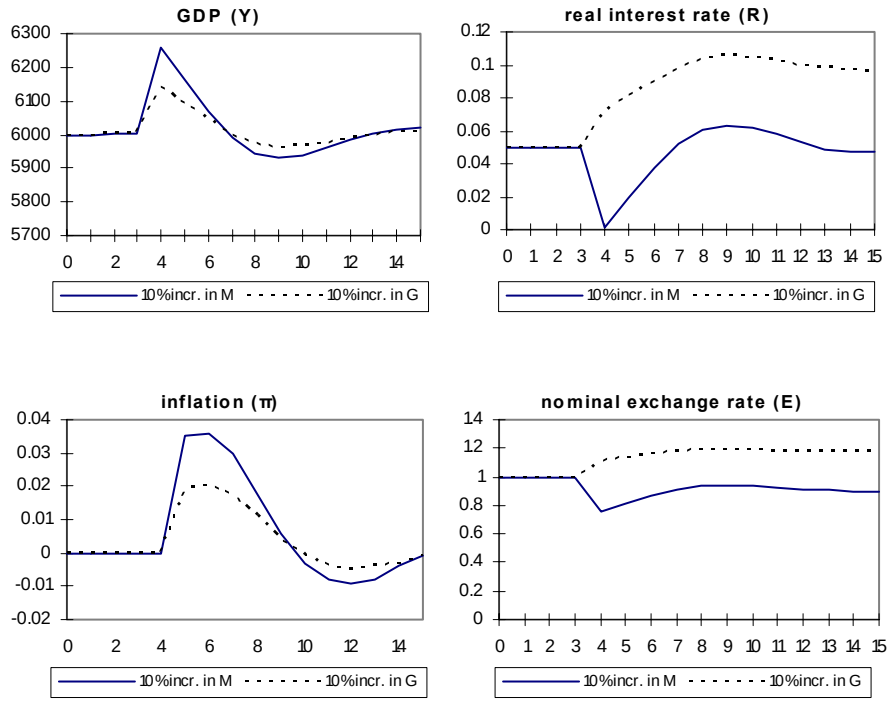


Figure 13.4 Effects of a 10% increase in the Money Supply and in Gov. Expenditure

We can observe how, as expected, the change in money supply has short-run but no long-run real effects, while the change in government expenditure has short and long-run real effects. We can also see how the trajectories to the new equilibrium positions are oscillatory, with temporary over and under-shooting of the final equilibrium positions.

The essential elements of the function of monetary policy can be seen in the results in Fig. 4. Consider the case where money supply is increased by 10% as is shown in the solid lines. This has the effect at first of decreasing the interest rate as in shown in the upper right diagram. The decrease in the interest rate in turn causes an increase in investment and therefore GDP as shown in the upper left hand graph. As GDP increases above potential, inflation increases as shown in the bottom left hand graph. The increase

in inflation raises the price level and this has the effect of decreasing the real money supply in the money demand equation

$$M/P = kY - hR \,. \tag{5}$$

This in turn causes an increase in the real interest rate beginning in period 5 as shown in the upper right hand graph. The rise in the interest rate then decreases investment and therefore GDP begins to fall in period 5 as shown in the upper left hand graph. This oscillatory process continues until GDP returns to the potential GDP level and inflation returns to zero.

In the GAMS program `htsim.gms` you will also find ways of changing more policy or exogenous variables to perform other experiments. For example, you will be able to simulate a change in potential GDP, or a change in the foreign price level.[33] You may also want to change the tax rate, which in the program is defined as a scalar, or any other model parameter.[34]

Having learned how to perform model simulations, we can now move to the realm of optimal policy analysis. This analysis is, in a way, the reverse of simulation. Instead of determining the paths of the endogenous variables given values for the policy

---

[33] If you change the foreign price level, you will notice that the nominal exchange rate also changes in an opposite and neutralizing way so that nothing else happens. From Eq. (9) we know that the real exchange rate is determined by the interest rate. We also know that the domestic price level is sticky in the short run. Thus a change in the foreign price level has to be compensated by a change in the nominal exchange rate. You will observe a similar behavior, but in the long run, in the case of a change in the money supply. Since this change affects the domestic price level but not the real interest rate in the long run, thus the nominal exchange rate will change to compensate the change in the domestic price level. Only in the case of a permanent change in the real interest rate (i.e. due to a change in government spending) will the nominal exchange rate and the domestic price level not move in a compensatory way.

[34] Hall and Taylor's textbook comes with a "black box" software named Macrosolve which allows you to perform experiments with the model changing some policy or exogenous variables. The GAMS program presented in this chapter replicates many results from Macrosolve. A change in the tax rate, since it is a model parameter, will change the steady-state solution of the model, as would be the case with any other model parameter such as the marginal propensity to consume, etc. However, for the particular experiment of changing the tax rate, Macrosolve gives steady-state invariant results. Our GAMS program doesn't. Thus, for that particular experiment, in case you wish to compare results, you will find that they differ. Notice that there is nothing wrong in one case or the other, just two different simulation methods.

variables, we now want to determine the optimal path for the policy variables given target paths and relative weights for target variables. This can easily be done by adding a loss function as an extra equation to the model and by redefining the policy variables of interest as endogenous variables. For example, in the GAMS statement above, we can substitute the following quadratic loss function for the previous `JD` equation and the `Loss` variable for the previous `J` variable, i.e.

```
eqLoss..  Loss  =E=  0.5  *  sum(t,  Wy * POWER((Y(t)-Ytar(t)), 2 )
                     +  Wp * POWER((P(t)-Ptar(t)), 2 );
```

where `Ytar` and `Ptar` are pre-specified target values for output and the price level and where `Wy` and `Wp` are weights on the deviations from target values of output and the price level respectively.

Since the variables entering the loss function (GDP and the price level) are measured in different units, it is convenient to impose some normalization on the weights. For instance, if `Ytar` is 6000 and `Ptar` is 1, then to equally penalize deviations from target we could set `Wy` equal to 1 and then obtain the corresponding normalized `Wp` as:

$$Wp = 6000^2 / 1^2 = 3600000.$$

Then, if we decide to penalize deviations from `Ytar` twice as much as for deviations from `Ptar`, we will choose `Wy = 2` and `Wp = 3600000`, or `Wy = 1` and `Wp = 1800000`, etc. For a full discussion of weighting procedures see Park (1997).

If we now redefine, for example, the money supply M(t) as an endogenous variable and we ask GAMS to solve the model minimizing the variable "Loss", we will obtain the corresponding optimal path for M(t). This is a typical and basic experiment in policy analysis. However, this analysis can be made more sophisticated in a variety of ways, for example by introducing stochastic elements and learning mechanisms. To do so, it may be convenient to move from GAMS to a more specialized software such as Duali. We will do that later in this book.

3. Experiments

In this chapter we simulated the effects of permanent changes in the money supply and in government expenditure. You may want to simulate temporary changes, that is, changes that last for only a few periods. To be acquainted with the dynamics of

the Hall and Taylor model, you should continue performing simulations of shocks to the model exogenous variables, i.e. potential GDP or the foreign price level, asking yourself if the observed effects make economic sense.

You may want to expand the model allowing for shocks to the domestic price level. This price shock may have different sources: changes in the price of an input to the economy (i.e. oil), a wage increase passed on by firms in the form of increased prices, etc. You can represent it as an exogenous variable *Z* added to Eq. (7) so that it becomes

$$\pi = \pi^e + f\left\{\left(Y_{-1} - Y_N\right) / Y_N\right\} + Z$$

Thus, this shock will be a shift factor in the short-run aggregate supply or horizontal price line. Notice that to properly introduce this new variable in the GAMS program you will have to define it as a parameter in the same fashion as we did potential GDP or the foreign price level and add it to the corresponding equation. You may want to try experiments in which this variable changes only temporarily. Notice also that this variable will be implicitly defined in percentage changes and not in levels.

Also, you may try to introduce changes in the model policy variables in order to counteract shocks to exogenous variables to bring the economy back to the initial equilibrium position, particularly in connection with the values of real variables. This is a rudimentary but useful way of undertaking policy analysis. Finally, you may want to perform a more sophisticated policy analysis shocking the economy with diverse shocks and working with a loss function as suggested at the end of this chapter, or you may decide to move on to the Macroeconomics in Duali chapter in this book where that kind of analysis is performed with a more specialized software.

Appendix 13A

Hall and Taylor in GAMS

```
$TITLE   htsim:   HALL-TAYLOR SIMULATION
OPTION SYSOUT = OFF;
OPTION LIMROW = 7;
OPTION LIMCOL = 0;
OPTION SOLPRINT = OFF;
$OFFSYMXREF OFFSYMLIST
*************************************************************************
* SECTION 1 :  DEFINITION OF PARAMETER VALUES FOR THE ORIGINAL
*                          NONLINEAR  HALL-TAYLOR MODEL
*************************************************************************
SCALARS
a       minimum consumption              / 220 /
b       marg prop to consume             / 0.7754 /
d       interest elast of invest.        / 2000 /
ee      maximum investment               / 1000 /
f       coeff. on excess aggr dem.       / 0.8 /
gg      maximum net exports              / 600 /
h       interest elast  of mon  dem.     / 1000 /
k       income elast  of money dem.      / 0.1583 /
mm      income elast  of net exp         / 0.1 /
n       real ex  rate elast  of net exp  / 100 /
q       constant                         / 0.75 /
tax     tax rate                         / 0.1875 /
v       constant                         / 5 /
alpha   coeff. on 1 lagged inflation     / 0.4 /
beta    coeff. on 2 lagged inflation     / 0.2 /
mu      elast. of empl. wrt GDP          / 0.33 /  ;

*************************************************************************
*  SECTION 2:     DEFINITION OF TEMPORAL HORIZON FOR SIMILATION
*************************************************************************
* If you change the extension of the horizon, make the necessary
* adjustments in the section of shocks' definition  (Section 3)

SETS  T    EXTENDED HORIZON     / 0*15 /
       T0(T)  PERIOD ZERO
       T1(T)  PERIOD ONE ;

       T0(T) = YES$(ORD(T) EQ 1);
       T1(T) = YES$(ORD(T) EQ 2);
       DISPLAY T0, T1;
```

```
*  SECTION  3 : DEFINITION OF CHANGES IN POLICY AND EXOGENOUS VARIABLES
*************************************************************************
PARAMETERS
*  definition of policy and exogenous variables (in percentage changes)
Mper(T)        money stock  (in % change)
Gper(T)        Gov. expenditure (in % change)
Ynper(T)       potential GDP (in % change)
Pwper(T)       foreign prices (in % change)
*  definition of policy and exogenous variables (in levels)
M(T)     money stock (in levels)
G(T)     Gov. expenditure (in levels)
Yn(T)    potential  GDP (in levels)
Pw(T)    foreign prices (in levels)  ;
*  default values for policy and exogenous variables
Mper(T) = 0 ;   Gper(T) = 0 ;  Ynper(T) = 0 ;  Pwper(T) = 0 ;
M(T) = 900 ;    G(T) = 1200 ;  Yn(T) = 6000 ;  Pw(T) = 1 ;
*****************************
* CHANGE IN MONEY SUPPLY
*****************************
SETS
TS1(T) periods for shock 1    / 4*15 / ;
Mper(TS1) = 0.0 ;
*************************************
* CHANGE IN GOVERNMENT EXPENDITURE
*************************************
SETS
TS2(T) periods for shock 2    / 4*15 / ;
Gper(TS2) = 0.0;
*************************************************************
*  CHANGE IN POTENTIAL GNP (notice that the natural rate of
*                           unemployment remains the same)
*************************************************************
SETS
TS3(T) periods for shock 3    / 4*15 / ;
Ynper(TS3) = 0.0;
***************************
*  CHANGE IN FOREIGN PRICES
***************************
SETS
TS4(T) periods for shock 4    / 4*15 / ;
Pwper(TS4) = 0.0;
*  Transformation of shocks in % changes into shocks in levels
M(TS1) =   900 * (1 + Mper(TS1)) ;
G(TS2) =  1200 * (1 + Gper(TS2)) ;
Yn(TS3) = 6000 * (1 + Ynper(TS3)) ;
Pw(TS4) =    1 * (1 + Pwper(TS4)) ;
```

```
* reporting policy and exogenous variables values
PARAMETER REPORTEX  POLICY AND EXOGENOUS VARIABLES VALUES;
  REPORTEX(T,"Money") = M(T);
  REPORTEX(T,"Gov. Exp.") = G(T);
  REPORTEX(T,"Pot. GDP") = Yn(T);
  REPORTEX(T,"Fgn Price") = Pw(T);

*************************************************************************
* SECTION 4:  COMPUTATION OF SOLUTION
*************************************************************************
PARAMETERS
Un(T)          natural rate of unemployment ;
Un(T) = 0.05 ;

VARIABLES
Y(T)           gdp
Yd(T)          disposable income
C(T)           consumption
I(T)           investment
R(T)           interest rate
P(T)           price level
pi(T)          inflation rate
piex(T)        expected inflation rate
E(T)           nominal exchange rate
X(T)           net exports
Gd(T)          government deficit
U(T)           unemployment rate
J              performance index

EQUATIONS
eq1(T)      gdp identity
eq2(T)      disposable income
eq3(T)      consumption
eq4(T)      investment
eq5(T)      money demand
eq6(T)      expected inflation
eq7(T)      inflation rate
eq8(T)      price level
eq9(T)      real exchange rate
eq10(T)     net exports
eq11(T)     government deficit
eq12(T)     unemployment rate
JD          performance index ;
```

```
JD..          J =E= 0 ;
eq1(t+2)..    Y(t+2)    =E=   C(t+2) + I(t+2) + G(t+2) + X(t+2) ;

eq2(t+2)..    Yd(t+2)   =E=    (1 - tax) * Y(t+2) ;

eq3(t+2)..    C(t+2)    =E=    a + b * Yd(t+2) ;

eq4(t+2)..    I(t+2)    =E=    ee - d * R(t+2) ;

eq5(t+2)..    M(t+2) / P(t+2)  =E=   k * Y(t+2) - h * R(t+2) ;

eq6(t+2)..    piex(t+2)=E=    alpha * pi(t+1) + beta * pi(t) ;

eq7(t+2)..    pi(t+2)   =E=    piex(t+2) + f*(Y(t+1)-Yn(t+2))/Yn(t+2) ;

eq8(t+2)..    P(t+2)    =E=    P(t+1) * (1 + pi(t+2)) ;

eq9(t+2)..    E(t+2) * P(t+2) / Pw(t+2)  =E=  q  + v * R(t+2) ;

eq10(t+2)..   X(t+2)    =E=   gg - mm*Y(t+2) - n*(E(t+2)*P(t+2)/Pw(t+2));

eq11(t+2)..   Gd(t+2)   =E=    G(t+2) - tax * Y(t+2) ;
eq12(t+2)..   U(t+2)    =E=    Un(t+2) - mu*(Y(t+2)-Yn(t+2))/Yn(t+2) ;

*************************************************************************
* In what follows, we assign initial variables' values and lower bounds
* WARNING: The order of declaration of assignments is very important
*      Successive assignments to a same variable undo the previous ones
*************************************************************************

*  Guess of initial values for the solution algorithm.
*  Without them, the problem may be declared "infeasible"
*  That is, the algorithm will converge to a solution from some initial
*  positions but not from others
*  This is common in nonlinear problems

 R.L(T+2) = 0.09 ; Y.L(T+2) = 6500 ; E.L(T+2) = 1.2; C.L(T+2) = 4500 ;
 I.L(T+2) = 900 ;  X.L(T+2) = -100 ; Gd.L(T+2) = 75 ; U.L(T+2) = 0.07 ;
 Yd.L(T+2)= 4875 ; pi.L(T+2) = 0.1 ; piex.L(T+2)=0.2 ; P.L(T+2) = 1.1 ;

* lower bound for p, to avoid  division by zero
 P.LO(T+2) = 0.0001 ;

* fixing initial steady-state values for lagged endogenous variables
 P.FX(T1) = 1 ;  pi.FX(T0) = 0 ;  pi.FX(T1) = 0 ; Y.FX(T1) = 6000 ;
```

```
MODEL NONLDYN  /eq1, eq2, eq3, eq4, eq5, eq6,
                eq7, eq8, eq9, eq10, eq11, eq12, JD / ;

SOLVE NONLDYN MINIMIZING J USING NLP;

* Reporting solution values
PARAMETER REPORTS SOLUTION VALUES IN LEVELS;
  REPORTS(T,"GDP")          = Y.L(T);
  REPORTS(T,"Inflation")    = pi.L(T);
  REPORTS(T,"Int.Rate")     = R.L(T);
  REPORTS(T,"Exch.Rate")    = E.L(T);
  REPORTS(T,"Gov.Def")      = Gd.L(T);
  REPORTS(T,"Unemploy")     = U.L(T);

* Showing final results
DISPLAY REPORTEX;
DISPLAY REPORTS;
```

Chapter 14

Agent-based Model in MATLAB

Agent-based Computational Economics is one of the newer fields in economics. Agent-based models simulate the behavior of multiple heterogeneous agents interacting in a variety of ways. While the modeling of economic agents has a long tradition in economics, agent-based modeling departs from it in a number of ways. For example, when modeling a market economy, the standard neoclassical competitive general equilibrium approach usually assumes that agents have fixed preferences, perfect and complete information, no reproductive behavior, and also that trade is organized by a central auctioneer that given all agents preferences and endowments computes the set of equilibrium prices. Thus, agents are price-takers and do not engage in trade at prices other than those given by the central auctioneer. Also space, that is geography, is usually an absent dimension in that approach. In contrast, agent-based models allow agents to display a number of more realistic characteristics and behaviors, i.e. changing preferences, bounded rationality and memory, imperfect and incomplete information, and local trade - agents may interact with neighbors in a geographically defined space and prices emerge from these decentralized interactions.

In this chapter we will introduce a famous agent-based model known as the Sugarscape model, developed by Joshua M. Epstein and Robert Axtell (1996). This is a model designed to simulate a variety of social phenomena such as population dynamics, migration, interaction with the environment, trade, group formation, combat and transmission of culture. We will learn how to represent and simulate the simplest version of this model in MATLAB. To do this, the knowledge of basic MATLAB operations and data types - vectors and matrices, with the addition of data types named structures and cell arrays that we will explain below - will suffice. However, more sophisticated simulations may require the use of object oriented programming techniques, something also available in MATLAB - see “MATLAB Classes and Objects” in the “Programming and Data Types” section of the MATLAB help navigator - as well as in lower level object oriented programming languages such as C++, C# or Java.

1. The Sugarscape Model: Introduction

The version of the classic Sugarscape model that we use in this chapter can be thought of as two major cities located near one another like Dallas and Fort Worth in Texas or Minneapolis and St. Paul in Minnesota. There is an original distribution of stores of a certain type in this terrain; for example, coffee houses such as Starbucks or perhaps mailing and business services stores such as UPS Stores. The franchise owners at each location work with varying degrees of efficiency and thus have different costs. They thus require different levels of revenues in order to continue to make a profit. Their profit each period is added to their accumulated wealth; however, if this wealth goes to zero the franchise is shut down. The surviving franchise owners each period look around for a nearby location that would be more favorable and move the store if they find a higher revenue location. However, some of the franchise owners scout longer distances away from their present store than others.

More formally, the Sugarscape model consists of two main elements: a terrain where events unfold named "sugarscape", which contains the spatial distribution of a generalized resource named "sugar" which can be thought of as the customer potential or revenue level at that location. The agents have metabolism levels and must eat to survive. This metabolism may be thought of as the cost of running the business in each period. Thus the difference between the sugar that the agents obtain at their location in each period and their metabolism level is like the profit of the enterprise in each period. This profit is accumulated as wealth from period to period; however, if the wealth level goes to zero in any period the agent dies, i.e. goes out of business. Thus, the agents are characterized by a set of fixed states (genetic characteristics such as metabolism and length of vision) and variable states (such as location and wealth) and move around the sugarscape following simple rules of behavior.

The sugarscape is represented by a two-dimensional coordinate grid or lattice. At every point of the grid given by the coordinates (x,y) there is a sugar level. Thus, we can easily represent the sugarscape in MATLAB by means of a matrix. For example, if we want to create and display a (50x50) sugarscape with a level of sugar equal to 4 units in the southwest quadrant and a level of 2 units elsewhere, we can do it with the following statements

```
for i = 1: 50;
      for j = 1: 50;
            if (i > 25 & j < 25)
                  s(i,j) = 4;
            else
                  s(i,j) = 2;
            end
      end
end
image(s);
```

In the statements above `image(s)` is a MATLAB function that displays the array `s`.

Figure 14.1 below shows the result, where the lighter region corresponds to the value 4 and the darker region corresponds to the value 2.

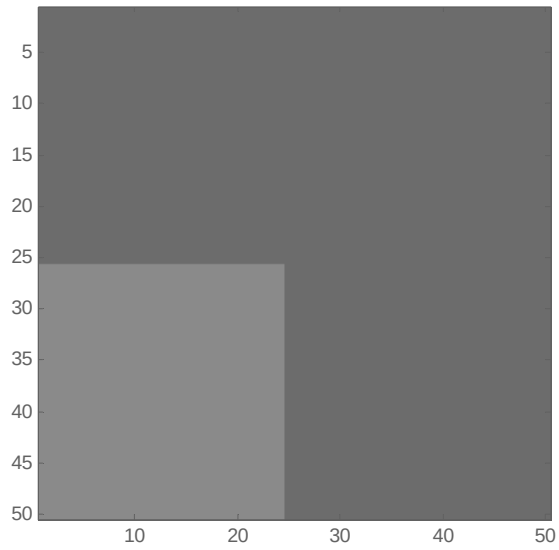


Figure 14.1 Sugarscape with Two Levels of Sugar

To represent agents, we can use another data type available in MATLAB called a structure. A structure is an array with "data containers" named "fields". These fields can contain any kind of data. For example, let's assume that every agent is characterized by two states: active, which signals if the agent is alive or not, with values equal to 1 and 0 respectively, and metabolism, that is the amount of sugar each agent has to eat per time period to survive. The statements

```
a_str.active = 1;
a_str.metabolism = 4;
```

create the simple 1x1 structure `a_str` containing two fields. If we use the statements

```
a_str(2).active = 1;
a_str(2).metabolism = 3;
```

then `a_str` becomes a 1x2 array with two fields. Let's assume that we want to create and display a random population of agents - say all those for whom the corresponding value from a [0,1] uniform distribution is lower than 0.2 - on a 50x50 grid. Also, we will assume that there can only be one agent on each location. We can achieve this with the following statements

```
for i = 1:50;
    for j = 1:50;
        if (rand < 0.2)
            a_str(i,j).active = 1; %put an agent on this location
            a_str(i,j).metabolism = 3;
        else
            a_str(i,j).active = 0; %keep this location empty
            a_str(i,j).metabolism = 0;
        end
    end
end
```

With these statements we can create a structure with 2,500 elements, each with two fields.

If we want to display the location of every agent on the grid, we can do it with the following statements, where we transfer the elements of the field `active` into the `a` matrix, and where the MATLAB function `spy(a)` displays all the nonzero elements in matrix `a`.

```
for i = 1:50;
    for j = 1:50;
      a(i,j) = a_str(i,j).active;
    end
end
spy(a);
```

The result, with a number of agents equal to 474, is shown in Figure 14.2 below, where `nz` means the number of non-zero elements.

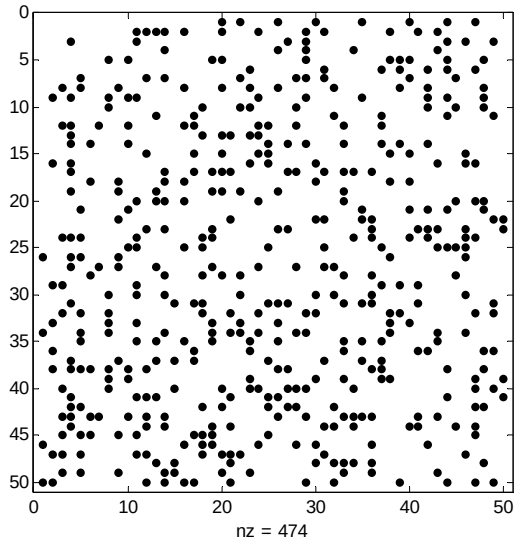


Figure 14.2 Agents Locations

Now that we have introduced the basic building blocks of the Sugarscape model and its MATLAB representation, we can move on to a more detailed presentation.

2. The Sugarscape Model

Next we present a more complex topography for the sugarscape and also more complex agent characteristics. We will also define rules that will govern the autonomous growth of sugar in the sugarscape and the movement of the agents on it.

We will assume that the sugarscape is characterized by two mountains of sugar, one in the southeast portion of the grid, and the other in the northwest, and that these two mountains are symmetric. Thus, for a 50x50 grid, we will assume that one peak of the sugarscape is approximately on the (0.75 * 50, 0.25 * 50) coordinate, while the other is on the (0.25 * 50, 0.75 * 50) coordinate. From the peaks down, the level of sugar at each location will follow decreasing paths.

We will also specify a very simple growback rule for the sugarscape:

Sugarscape rule G_∞ *:* Grow back to full capacity immediately.

Thus, at each run of the model, the level of sugar grows back to its initial level. The symbol G_{∞} here is a fancy way to specify how rapidly the amount of sugar (revenue) grows back in each time period. Epstein and Axtell (1996) use a variety of such rules.

We will also assume that the sugarscape is what in geometry is know as a Torus, or in a more familiar way, that it corresponds to the surface of a donut. This means, for example, that an agent moving to the south on column 6, after reaching row 50 will appear on the sugarscape from the north in the coordinate (1,6), and an agent moving to the east on row 6, after reaching column 50 will appear on the sugarscape from the west on the coordinate (6,1). Analogous patterns will be followed by agents moving north or west.

Turning now to the agents, we will assume that each agent has four characteristics, two of them fixed and the other two variable. The fixed ones are metabolism - the amount of sugar the agent has to consume at each time period to stay alive - and vision - the number of sites in the sugarscape each agent can see. We will assume that agents can see only in four directions: north, south, east and west. Thus, they can not see in diagonal directions. The level of vision is the maximum number of sites each agent can see in a given direction. Metabolism and vision are genetic characteristics randomly distributed among agents.

The variable characteristics of agents are location on the sugarscape and wealth, with the later understood as the agents stock of sugar. We will assume that agents are randomly born around the sugarscape at the beginning of the simulation. Each agent will start its life with a level of wealth equal to the level of sugar in the sugarscape location were it was born.

We will specify a rule that will govern the behavior of each agent on the sugarscape:

Agent movement rule M:

- Look out as far as vision permits in the four principal directions and identify the unoccupied site(s) having the most sugar
- If the greatest sugar value appears on multiple sites then select the nearest one
- Move to this site
- Collect all the sugar at this new position

Once sugar is collected, the agent's wealth is incremented by the sugar collected and decremented by its metabolic rate. An agent lives forever, unless its wealth is below its metabolic rate. In this case, it dies and is removed from the sugarscape. In principle,

all agents should apply this rule simultaneously. However, since the simulation is run on a serial computer, only one agent will be active at any instant. In this case, it is recommended to randomize agents' order of movement, and we will do this in the MATLAB code. We will also randomize step one of the rule, that is, the order in which each agent searches the four directions.

Having presented the building blocks of the simplest version of the Sugarscape model, we now turn to its MATLAB representation.

3. The Sugarscape Model in MATLAB

The MATLAB representation consists of a main program named `sugarscape1.m` and a number of functions, all of which are available from the book web site. Below is the code of the main program.

```
%Initialize model parameters
nruns = 6;
size = 50; %even number
metabolismv = 4;
visionv = 6; %set always smaller than size
maxsugar = 20;

%Initialize sugarscape and display
s = initsugarscape(nruns, size, maxsugar);

%Initialize agents population
a_str = initagents(size, s, visionv, metabolismv);


%Main loop (runs)
for runs = 1:nruns;
    % Display agents' locations
    dispagentloc(a_str, size, nruns, runs);

    % Select agents in a random order and move around the sugarscape  %
    % following rule M
     for i = randperm(size);
        for j = randperm(size);
            if (a_str(i,j).active == 1) %is there an agent on this
                                        %location?
                %Agent explores sugarscape in random directions and
                  %selects best location
                temps = s(i,j);
                tempi = i;
                tempj = j;
                for k = a_str(i,j).vision : -1 : 1;
                    [temps, tempi, tempj] =
                     see(i,j,k,a_str,s,size,temps,tempi,tempj);
                end
                %Agent moves to best location, updates sugar stock and
                 %eats sugar
                a_str = moveagent(a_str, s, i, j, temps, tempi, tempj);
            end       % if
        end           % for j
    end               % for i
end                   % for runs
```

The program begins with the initialization of the model parameters - the number of runs, the size of the sugarscape, the maximum value of metabolism and vision of the agents, and the maximum level of sugar in the sugarscape. Then follows a call to the

function named `initsugarscape`, which will return a matrix named `s` containing the sugar levels in the sugarscape. Next a call to the function `initagents` returns the data structure `a_str` which will contain the agents' population.

Then follows the main loop of the program corresponding to the number of runs - each run represents a time period - of the simulation. At each pass of the loop, the locations of the agents on the sugarscape are displayed as a way of visualizing their movements. This is achieved by calling the function `dispagentloc`.

Then each agent, in a random order, explores the sugarscape, selects the best location, updates its wealth and eats sugar to survive. This section of the program begins with the following statements.

```
for i = randperm(size);
      for j = randperm(size);
```

The `randperm(n)` function performs a random permutation of the elements of the set (1,2,…,n). Thus the `randperm(size)` MATLAB function creates a vector with a number of elements equal to `size` and performs a random permutation of those elements. Thus, once the two `for` loops - one for `i` and the other for `j` - are completed, the whole population of agents will have moved but in a random order. The conditional

```
if (a_str(i,j).active == 1) %is there an agent on this location?
```

checks if there is an active agent in the `(i,j)` location being examined, where a 1 in the field `active` of the agent data structure denotes that there is an agent, while a 0 denotes the opposite. Then, if there is an active agent in the location, the program proceeds to apply the agent's rule of movement, while if that is not the case it proceed to examine another location looking for an active agent. The agent's rule of movement is implemented with the statements below

```
%Agent explores sugarscape in random directions and
%selects best location
temps = s(i,j);
tempi = i;
tempj = j;

for k = a_str(i,j).vision : -1 : 1;
      [temps, tempi, tempj] =
      see(i,j,k,a_str,s,size,temps,tempi,tempj);
end

%Agent moves to best location, updates sugar stock and
%eats sugar
a_str = moveagent(a_str, s, i, j, temps, tempi, tempj);
```

The statements begin with the setting of three temporary variables. The variable `temps` contains the level of sugar in the agent's current location, while `tempi` and `tempj` contain the location's coordinates. Then follows a loop that goes from the agent's maximum level of vision to 1, in decrements of one unit. At each pass of this loop, the function `see` is called. This function will see around the agent's neighborhood in the north, south, east and west directions, from the farthest position the agent can see to its immediate surroundings, and will return the maximum level of sugar in the variable `temps` and its location coordinates in the variables `tempi` and `tempj` respectively. Finally, once the loop is completed, the function `moveagent` is called to move the agent to the new location and to update its stock of wealth.

From this overview of the main program we turn next to descriptions of the functions.

3. Functions

3.1 Initsugarscape

The "initsugarscape" function initializes the level of sugar at each location of the sugarscape. To better understand the procedure used, we will begin with simpler examples. Suppose that we want to generate an 11x11sugarscape `s1` with a single mountain with a peak in the center. The corresponding statements are shown below, where `i` and `j` are the matrix coordinates varying from 1 to 11. The vectors `x` and `y` are two identical eleven-element vectors containing the values [-5 -4 -3 -2 -1 0 1 2 3 4 5].

```
%Generate sugarscape with one peak in the center
x = -5:5;
y = -5:5;
maxsugar = 20;
for i = 1:11;
    for j = 1:11;
        if (x(i) == 0 & y(j) == 0)
            s1(i,j) = maxsugar;
        else
            s1(i,j) = maxsugar / (abs(x(i)) + abs(y(j)));
       end
    end
end
```

The value of each element in the `s1` matrix is computed dividing the given maximum level of sugar by the sum of the absolute value of the corresponding elements in the `x` and `y` vectors as shown below:

```
s1(i,j) = maxsugar / (abs(x(i)) + abs(y(j)));
```

where `abs` is the absolute value. The peak of the mountain will be where the corresponding elements of the `x` and `y` vectors equal zero. Thus, the value of `s1(6,6)`, which will be located at the center of the sugarscape, will be equal to `maxsugar` - making a minor adjustment to avoid the division by zero. And the values on the corners - i.e. `s1(1,1)` - will be equal to (`maxsugar/10`). All the other values would be, in a decreasing order, between `maxsugar` and (`maxsugar/10`) as shown in Figure 14.3 below.

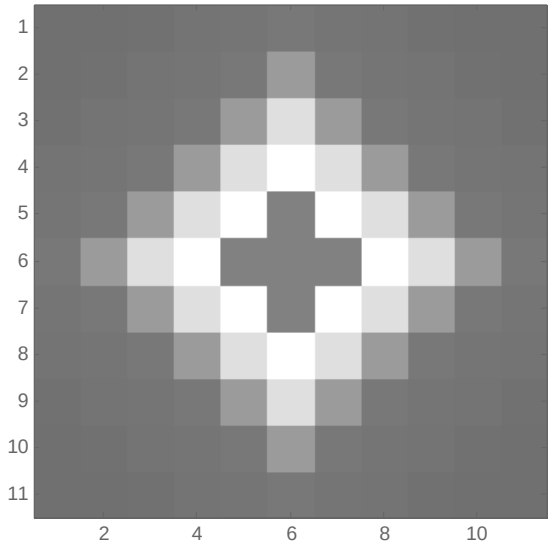


Figure 14.3 Sugarscape with a Center Peak

Now, if we want to generate a sugarscape with a peak in the southeast instead of the center, the values of `x` and `y` should be shifted to

x = [-9 -8 -7 -6 -5 -4 -3 -2 -1 0 1]

and

y = [-3 -2 -1 0 1 2 3 4 5 6 7].

In this case, the peak of the sugarscape will be in the `s1(10,4)` location, as shown in Figure 14.4 below.

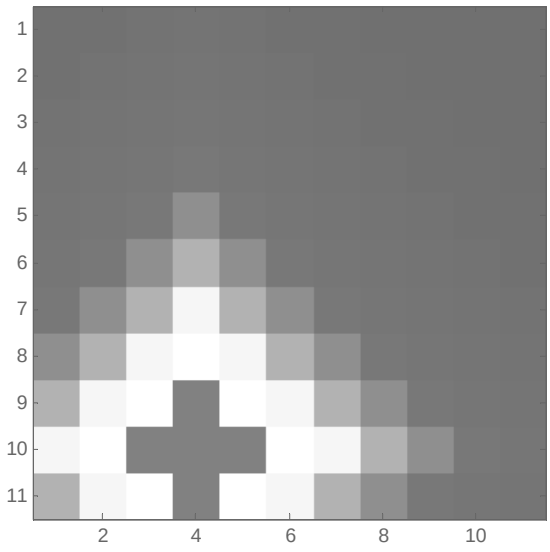


Figure 14.4 Sugarscape with a South-West Peak

The `initsugarscape` function initializes the level of sugar at each location of the sugarscape. This particular function will generate a topography characterized by two mountains of sugar, one in the southwest portion of the grid, and the other in the northeast. These two mountains are symmetric. From the peaks down, the level of sugar will follow decreasing paths. The function code is available in file `initsugarscape.m`,

This function begins by generating a sugarscape `s1` containing a single peak in the southwest. To do so, the "Generate sugarscape with one southwest peak" section of the function, reproduced below, applies a similar procedure to the one just described.

```
%Generate sugarscape with one south west peak
x = -ceil(0.75*size) : size-ceil(0.75*size)-1;
y = -ceil(0.25*size) : size-ceil(0.25*size)-1;

for i = 1:size;
    for j = 1:size;
        if (x(i) == 0 & y(j) == 0)
            s1(i,j) = maxsugar;
        else
            s1(i,j) = maxsugar / (abs(x(i)) + abs(y(j)));
       end
    end
end
```

For example, for a value of `size` equal to 50, it begins by generating a 50-element vector `x`. The statement

```
x = -ceil(0.75*size) : size - ceil(0.75*size) - 1;
```

is used to create a 50 element vector of integers as follows. The values in the vector begin at minus the ceiling of the product (0.75 * 50), i.e. the next integer above 37.5, namely -38. They end at the value (50 - 38 -1), i.e. 11. So x will be a 50 element vector with the values

$$[-38,-37,\cdots,-1,0,1,\cdots,10,11]$$

Thus, the value zero will be in the 39th position of the `x` vector. In a similar way the vector `y`, which goes from -13 to 36, is generated with the value zero in its 14th position.

After doing this, each element of the sugarscape matrix `s1` is generated. The result will be a sugarscape with a peak in the `s1(39,14)` location, i.e. in the southwest corner of the array.

Once the first mountain is generated, a symmetric one is obtained by transposing the matrix `s1` with the statement

```
s2 = s1';
```

Then, the statement

```
s = s1 + s2;
```

generates the two-peak sugarscape. The following two statements

```
maxrow = max(s);
max(maxrow)
```

compute the row containing the maximum value in the matrix `s` and print the maximum value in this row. This may seem redundant, since we set the parameter `maxsugar` at the beginning of the program. That value is indeed the maximum for the peaks in `s1` and `s2`. But the peaks in `s` will be a bit higher since to each original peak we will be adding the value of the corresponding cell in the symmetric matrix, which will be a low value given its distance from the peak.

The final statements below display the image of the sugarscape shown in Fig. 14.5.

```
figure(1);
imagesc(s);
axis square;
```

The statement `figure(1)` generates a figure where an image will be displayed. The statement `imagesc(s)` scales the data in matrix `s` to the full range of colors and displays the corresponding image of the sugarscape matrix `s`. Finally, the statement `axis square` makes the image square. The result is the figure with two centers of economic activity as shown below.

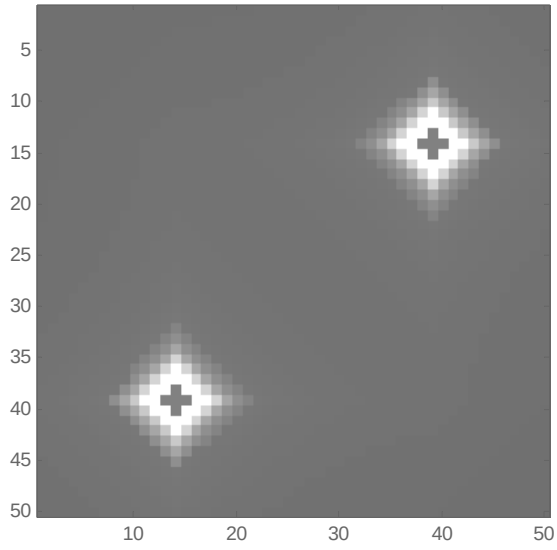


Figure 14.5 Two-peak Sugarscape

Next we turn from the code for the sugarscape to the code for the agents.

3.2 Initagents

The function "initagents" generates a random initial population of agents. Its code is shown below.

```
function a_str = initagents(size, s, visionv, metabolismv);
for i = 1:size;
    for j = 1:size;
        if (rand < 0.2)
            a_str(i,j).active = 1; %put an agent on this location
            a_str(i,j).metabolism = ceil(rand * metabolismv);
            a_str(i,j).vision = ceil(rand * visionv);
            a_str(i,j).wealth = s(i,j);
        else
            a_str(i,j).active = 0; %keep this location empty
            a_str(i,j).metabolism = 0;
            a_str(i,j).vision = 0;
            a_str(i,j).wealth = 0;
        end
    end
end
```

The information about agents is stored in the data structure `a_str` with four fields. The field `active` contains a 1 or 0 depending of the situation of the agent in a specific location (active, that is alive; or inactive, that is dead). A location with an inactive agent is treated in the main program and other functions as an empty location. If the values generated by the uniform distribution MATLAB function `rand` are below 0.2, an agent is born.

The fields `metabolism` and `vision` contain the corresponding integers randomly distributed between 1 and the maximum level of each characteristic. The MATLAB function `ceil` is used to round the randomly created `vision` and `metabolism` variables up to the next integer. The field `wealth` is initialized as equal to the amount of sugar in the location of the sugarscape where the agent was born.

3.3 Dispagentloc (display agent location)

This simple function transforms the field `agent` from the agents data structure into a matrix named `a` and displays agents' locations, since MATLAB does not allows one to display that field directly. The code of the function is shown below.

```
function a = dispagentloc(a_str, size, nruns, runs);
for i = 1:size;
    for j = 1:size;
      a(i,j) = a_str(i,j).active;
    end
end
figure(2);
subplot(ceil(sqrt(nruns)),ceil(sqrt(nruns)),runs), spy(a);
axis square;
```

The statement `figure(2)` tells MATLAB to display a second figure with the agent's locations - remember that a first figure was created before to display the sugarscape.

Consider next the line of code

```
subplot(ceil(sqrt(nruns)),ceil(sqrt(nruns)),runs), spy(s);
```

and notice that this one line contains two separate MATLAB statements, i.e. the function calls

```
subplot()
```

and

```
spy()
```

The call to `subplot` divides the window into a number of panes and the call to `spy` plots the active pane. These statements thus allow us to display multiple images in a single figure such as the images of agents' locations in successive runs of the program. The MATLAB function

```
subplot(m,n,p);
```

creates an axes in the pth pane of a figure divided into an m-by-n matrix of rectangular panes. For example, if we set the number of runs parameter in the main program equal to 8, then the statement

```
subplot(ceil(sqrt(nruns)),ceil(sqrt(nruns)),nruns), spy(s);
```

where `ceil(sqrt(nruns)` is the ceiling (i.e. the integer above) the square root of the number of runs, will divide the figure (window) into a matrix with 3 rows and 3 columns of panes to accommodate the images of the agent's locations in successive runs.

3.4 See and Neighbor

The `see` and `neighbor` functions explore the neighborhood an agent can see according to its level of vision in four directions - north, south, east and west - each direction selected in a random order. Remember that the location coordinates of the agent are given by `(i,j)` and that the agent's level of vision is equal to `k`. For each integer between `k` and `1` - that is, going from the outermost part of the neighborhood to its center - the function will check the level of sugar in each of the four directions. Every time the level of sugar in a location being examined is greater than the level of sugar in the agent's location, the level and coordinates of the higher value found will be stored in the temporary variables `temps`, `tempi` and `tempj` respectively. Thus, at the end of the exploration, these variables will contain the highest level of sugar found and its location.

Imagine that we begin by exploring the neighborhood in the south direction for a level of vision equal to `k` and from the location `(i,j)`. Thus, we want to examine the location `(i+k,j)`. If `(i+k <= size)`, where size is the dimension of the sugarscape, there is no problem. However if `(i+k > size)`, we have to remember that in Section 2 above we define the sugarscape as a Torus. Then, in this case the location to be examined will be `(i+k-size, j)`. For example, if we start from the location `(48,2)` with `k = 6`, then the location to be examined will be `(4,2)`. Thus, to summarize, we could write the following pseudo code, where `neighbor` will be a function that will check the level of sugar in the location `(u,v)`.

```
if (i + k > size)
    u = i + k - size;
    v = j;
    neighbor(u,v);
else
    u = i + k;
    v = j;
    neighbor(u,v);
end
```

Now, in the case when we want to examine the north direction, the code should be

```
if (i - k < 1) %or equivalently if(k - i > -1)
    u = i - k + size;
    v = j;
    neighbor(u,v);
else
    u = i - k;
    v = j;
    neighbor(u,v);
end
```

Analogous codes could be written for the cases of the east and west directions. However, we want to write a general code encompassing all the four cases. That is, something of the form

```
if ( (1) > (2) )
    u = (3);
    v = (4);
    neighbor(u,v);
else
    u = (5);
    v = (6);
    neighbor(u,v);
end
```

To do so, we proceed as follows. We define the following four vectors, each with six elements:

```
south = [i+k  size  i+k-size  j        i+k  j];
north = [k-i  -1    i-k+size  j        i-k  j];
east  = [j+k  size  i         j+k-size  i   j+k];
west  = [k-j  -1    i         j-k+size  i   j-k];
```

Next we make use of a MATLAB object named “cell array”. A cell array is an array whose elements are also arrays. For our case, think of it as a matrix whose elements are vectors instead of numbers. The following statements create a cell array of dimension 1x4 whose elements are the vectors south, north, east and west. Notice that the indexes of a cell array are between braces.

```
c{1} = south; c{2} = north; c{3} = east; c{4} = west;
```

Now, for example, if we want to access the third element of the north vector, we can do it using a double indexing notation such as

```
c{2}(3);
```

Then, a general code to explore the neighborhood of an agent, selecting four directions of search in a random manner, can be written as:

```
for m = randperm(4);
      if (c{m}(1) > c{m}(2))
        u = c{m}(3);
        v = c{m}(4);
        [temps, tempi, tempj] =
         neighbor(u,v,a_str,s,temps,tempi,tempj);
      else
        u = c{m}(5);
        v = c{m}(6);
        [temps, tempi, tempj] =
         neighbor(u,v,a_str,s,temps,tempi,tempj);
      end
end
```

To check this go through the south and then the north cases and you should get the same results as those shown above.

We turn now to explain the workings of the `neighbor` function, which is a very simple one. As can be seen in the code above this function receives as inputs, among other arguments, the variables `temps`, `tempi`, and `tempj` and returns the same variables as outputs. Remember that `temps` contains the level of sugar in a given location and `tempi` and `tempj` contain the coordinates of the location. The code of the `neighbor` function is shown below.

```
function [temps, tempi, tempj] =
neighbor(u,v,a_str,s,temps,tempi,tempj);

if (a_str(u,v).active == 0)
    if (s(u,v) >= temps)
        temps = s(u,v);
        tempi = u;
        tempj = v;
    end
end
```

Thus, the function first checks whether the `(u,v)` location is free so that an agent can move there. If that is the case, it checks to see whether or not the level of sugar in the `(u,v)` location of the sugarscape is greater than or equal to the one previously found and stored in the variable `temps`. If so, it puts the new level found in the `temps` variable, and its corresponding `(u,v)` coordinates in the variables `tempi` and `tempj`.

To conclude this section, we reproduce below the entire code of the `see` function.

```
function [temps, tempi, tempj] =
see(i,j,k,a_str,s,size,temps,tempi,tempj);

south = [i+k  size  i+k-size  j  i+k  j];
north = [k-i  -1  i-k+size  j  i-k  j];
east  = [j+k  size  i  j+k-size  i  j+k];
west  = [k-j  -1  i  j-k+size  i  j-k];

c{1} = south;  c{2} = north;  c{3} = east;  c{4} = west;

for m = randperm(4);
      if (c{m}(1) > c{m}(2))
        u = c{m}(3);
        v = c{m}(4);
        [temps, tempi, tempj] =
         neighbor(u,v,a_str,s,temps,tempi,tempj);
      else
        u = c{m}(5);
        v = c{m}(6);
        [temps, tempi, tempj] =
         neighbor(u,v,a_str,s,temps,tempi,tempj);
      end
end
```

3.5 Moveagent

Once the neighborhood of the agent has been examined, it is time to move the agent to the best location found, update its wealth and let it eat sugar. This is what the `moveagent` function shown below does.

```
function a_str = moveagent(a_str, s, i, j, temps, tempi, tempj);

if (temps > s(i,j))
    % Agent moves to best location and updates wealth
    a_str(tempi,tempj) = a_str(i,j);
    %Set old location to unoccupied
    a_str(i,j).active = 0;
    a_str(i,j).vision = 0;
    a_str(i,j).metabolism = 0;
    a_str(i,j).wealth = 0;
    % update wealth at new location
    a_str(tempi,tempj).wealth = a_str(tempi,tempj).wealth + temps -
                                a_str(tempi,tempj).metabolism;
    % if wealth is less than zero set location to unoccupied
    if (a_str(tempi,tempj).wealth <= 0)
      a_str(tempi,tempj).active = 0;
      a_str(tempi,tempj).vision = 0;
      a_str(tempi,tempj).metabolism = 0;
      a_str(tempi,tempj).wealth = 0;
    end
else
    % Agent stays in position and updates wealth
    a_str(i,j).wealth = a_str(i,j).wealth + temps -
                        a_str(i,j).metabolism;
    if (a_str(i,j).wealth <= 0)
      a_str(i,j).active = 0;
      a_str(i,j).vision = 0;
      a_str(i,j).metabolism = 0;
      a_str(i,j).wealth = 0;
    end
end
```

If a new and better location than the one previously occupied by the agent is found, that is, if the statement below is true

```
      if (temps > s(i,j))
```

then the agent moves to the new location whose coordinates are stored in the variables `tempi` and `tempj`. The old location is set to unoccupied, and the agent's wealth is updated adding to its previous wealth the amount of sugar found in the new location and subtracting the sugar to be consumed according to its metabolic rate. If the resulting level of wealth is less or equal than zero then the agent dies and all its fields are set to zero.

In the case that no better location was found, the agent stays into place, updates its wealth and eats sugar. Again, if the resulting level of wealth is less or equal to zero, the agent dies.

4. Results

We are now ready to analyze the behavior of the population of agents in the sugarscape given the topography, the growback rule G_∞ and the agents' rule of movement M. The agents' locations for six successive runs, for a maximum vision of 6 and a maximum metabolism equal to 4, are shown in Figure 14.6 below. The order of graphs corresponding to the successive runs goes from left to right then down to the next row.

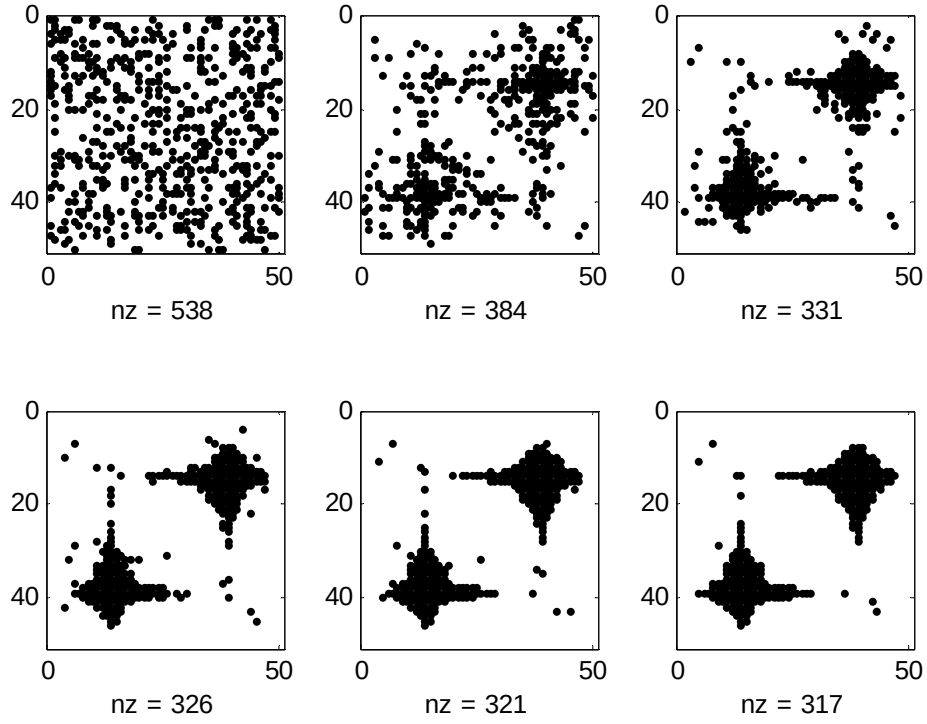


Figure 14.6 Agents' Locations for Six Runs

We can observe that in the first run there is a total population of 538 agents (nz means non-zero elements) randomly distributed on the sugarscape. As one would expect, during each run some agents die and others move toward the peaks of the sugarscape. For this experiment, the average metabolism of the population goes from 3.5 in the first run to 2 in the sixth run while the average vision goes from 3.5 to 3.8. Thus, as one should expect, lower metabolism and higher vision increase the chances of survival. We can see also that the population tends to reach a stable size and spatial configuration.

Figure 14.7 below shows the carrying capacity of the sugarscape - that is what population size the sugarscape can support - as a function of the maximum level of vision and metabolism of the agents. For each level of vision and metabolism, the average value of ten simulations of six runs each is presented. We can observe how a larger vision and a smaller metabolism tend to increase the carrying capacity of the sugarscape.

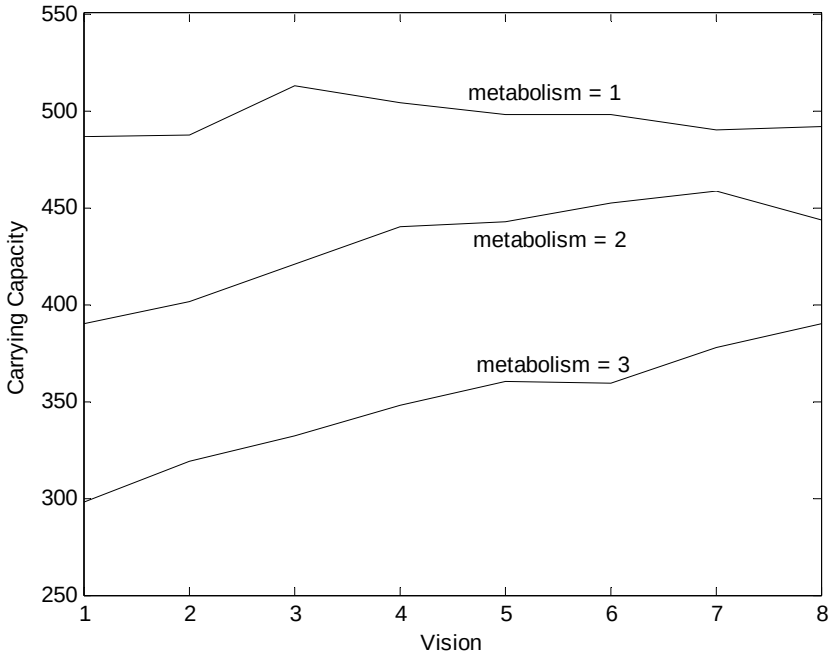


Figure 14.7 Carrying Capacity

5. Experiments

A simple experiment would be to add moving cost proportional to the distance moved. This will tend to slow down the convergence to the hilltop locations.

Also, the Sugarscape model can be extended in a number of ways so that many experiments of increasing grade of complexity can be performed. A first step in that direction would be to replace the Sugarscape rule G_∞ used above with the following one:

Sugarscape growback rule G_1*:* At each lattice position, sugar grows back at a rate of α units per time interval up to the capacity at that position.

To introduce this rule, you may want to start by transforming the sugarscape matrix `s` into a structure with two fields, one containing the capacity and the other the current level of sugar. Then, you can check how different growback rules affect the results.

You may also try to work with agents with finite lives, where their maximum age is a random integer drawn from a given interval [a,b]. Then, you may introduce an agent replacement rule such as the following one

Agent replacement rule $R_{[a,b]}$: When an agent dies it is replaced by an agent of age 0 having random genetic attributes, random position on the sugarscape, random initial endowment, and a maximum age randomly selected from the range [a,b].

Epstein and Axtell (1996) present a number of rules for pollution formation, agent mating, agent inheritance, trade, credit, etc., that can be implemented in the Sugarscape model. To learn about the specifics of these rules you are referred to their book.

6. Further Reading

For a comprehensive presentation of the Sugarscape model see Epstein and Axtell (1996). See also the web page of the Sugarscape model at the Brookings Institution at www.brook.edu/es/dynamics/sugarscape/default.htm . For an online guide to agent-based modeling see Axelrod and Tesfatsion (2004). For an approach to estimating agent based models see Gilli and Winker (2003).

For a recent conference keynote address on agent based modeling and an application to finance see LeBaron (2004). Also see his survey paper on agent based computational finance (LeBaron (2005)) which will appear in the Judd and Tesfatsion (2005) volume containing many state-of-the-art papers on agent based modeling. For a comprehensive site with resources on Agent-Based Computational Economics, see the web site developed by Leigh Tesfatsion at www.econ.iastate.edu/tesfatsi/ace.htm . For a review of agent-based modeling as an approach to economic theory, see Tesfatsion (2005).

Chapter 15

Global Warming in GAMS

The basic economics and chemistry of global warming are that an increase in output causes an increase in CO_2 emission which in turn causes an increase in the concentration of CO_2 in the atmosphere. This increase in CO_2 concentration permits the sun's rays to come into the earth's atmosphere but captures some of them as they are reflected back thereby increasing the temperature of the earth. The increased temperature results in a decrease in output. Several of the elements in this chain of causation are controversial; however this simple line of reasoning is a useful place to begin.

In this chapter we use the classic global warming model of Nordhaus (1992) to study the dynamics of global warming. A simple flowchart for that model, reflecting the discussion above, is shown in Figure 15.1.

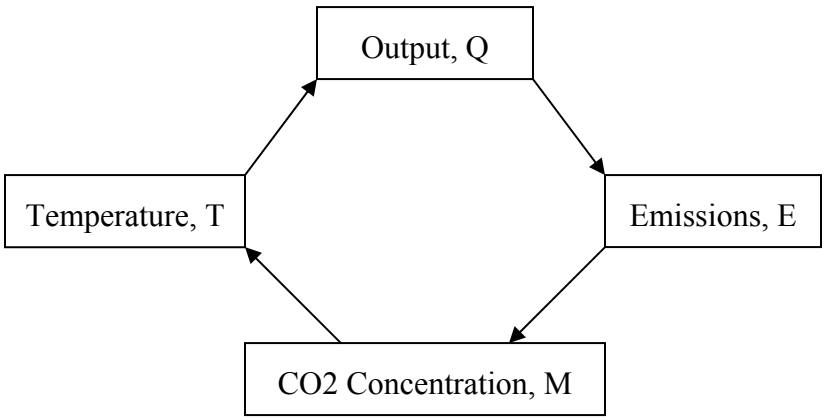


Figure 15.1 Basic Flowchart of Global Warming

Economic policy can be used in intervene in this cycle. The most common intervention is a "carbon" tax which raises the price of fossil fuels like coal, oil and natural gas and thereby decreases the effective emission of CO_2 and other greenhouse gases. This decreases the CO_2 concentration and therefore the temperature. This in turn tends to *increase* output. However the tax also decreases the efficiency of the economy, thereby providing a tendency to *decrease* output. This tradeoff is shown in Figure 15.2. Thus the basic structure of this dynamic model is one in which the economic externality is a *stock* variable, i.e. the CO_2 concentration, and the policy variable is used to control a *flow*, namely the CO_2 emissions.

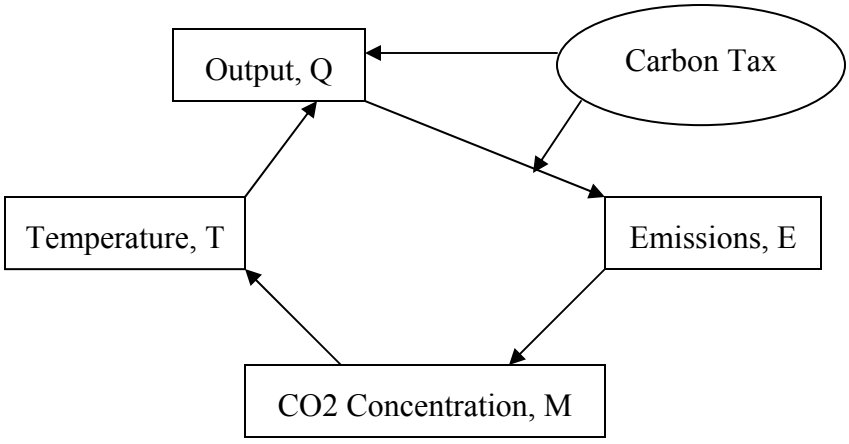


Figure 15.2 Policy Interventions with a Carbon Tax

The tradeoff was embedded by Nordhaus in a one-sector growth model, similar to the Excel growth model used earlier in this book, thereby creating an economic model of global warming. However, he developed the model in GAMS rather than in Excel as is discussed in the following sections. We begin with a discussion of the model in mathematics and then turn to a discussion of the model in GAMS.

1. The Mathematical Model

The best place to start is with the production function which is in the classic Cobb-Douglas form with output produced by capital and labor. This function is written

$$Q(t) = \Omega(t)\, A(t) K(t)^{\gamma} L(t)^{1-\gamma} \tag{1}$$

where

$Q(t)$ = output in period t

$\Omega(t)$ = climate impacts (see below)

$A(t)$ = technology in period t

$K(t)$ = capital in period t

$L(t)$ = labor force in period t

γ = elasticity of output with respect to capital

The unusual aspect of this production function is the presence of the Ω term which is used (1) to model the impact of temperature changes on output and (2) the efficiency-loss effects of the carbon tax. Also, as we will see later in more detail, in this model no distinction is made between labor force and population.

We will return to a discussion of the Ω term later; however for now lets move on to the effect of output on greenhouse gas emissions (mostly CO_2) that is modeled with the equation

$$E(t) = \left[1 - \mu(t)\right] \sigma(t) Q(t) \tag{2}$$

where

$E(t)$ = green house gas emissions

$\mu(t)$ = emission control rate – the fractional reduction of emissions

$\sigma(t)$ = ratio of greenhouse gas emissions to output

The μ variable is the percentage of greenhouse gas emissions which is prevented from entering the atmosphere. So it might be thought of as the action of devices to reduce the CO_2 in the smoke from the tall stacks of power plants or to sequester the carbon underground or underwater before it enters the atmosphere. Alternatively, it can be viewed as a proxy for a carbon tax which reduces the use of fossil fuels and thereby the effective emissions.

Next consider the effect of the emissions on the CO_2 concentration in the atmosphere, which is modeled with the equation

$$M(t) = \beta E(t) + (1-\delta_M) M(t-1) \tag{3}$$

where

$M(t)$ = CO_2 concentration relative to pre-industrial times

β = marginal atmospheric retention ratio

δ_M = rate of transfer from the rapidly mixing reservoirs to the deep ocean

The two parameters in this equation (β and δ_M) divide the non-intervention optimist from the intervention pessimist on global warming. The β parameter is the proportion of emissions that add to the CO_2 concentration in the atmosphere. The δ_M parameter is a measure of the atmosphere's ability to breakdown the CO_2. If δ_M is large, then the decay rate of CO_2 in the atmosphere is high and that mitigates the effect of higher emission rates. So the optimists like to believe that β is small and δ_M is large.

The increase in the atmospheric concentration of CO_2 in Eq. (3) in turn drives changes in temperature. This is done in two steps in the model. In the first step the increase in atmospheric concentration M increases the forcing term F in the equation

$$F(t) = 4.1\left[\frac{\log M(t)}{\frac{590}{\log 2}}\right] + FO(t) \tag{4}$$

where

$F(t)$ = forcing term of greenhouse gas concentration on temperature

$FO(t)$ = exogenous forcing from other greenhouse gases

This first term on the right hand side of Eq. (4) models the effect of the CO_2 concentration on the forcing term. The equation also includes a separate exogenous term for the effects of all other greenhouse gases on the forcing term.

The forcing term then influences the temperature. However, temperature is broken into two separate variables in this model – (1) the temperature of the atmosphere and upper oceans and (2) the temperature of the deep oceans. For simplicity of exposition, we will refer to the first of these two as just the temperature of the

atmosphere, though the reader should keep in mind that it is actually the temperature of the atmosphere and the upper oceans.

The forcing term F drives the temperature of the atmosphere and also the two temperatures have feedback effects on one another. The expression for the temperature of the atmosphere is

$$(5) \qquad T_1(t) = T_1(t-1) + \left(\frac{1}{R_1}\right)\left\{F(t) - \lambda T_1(t-1) - \left(\frac{R_2}{\tau_2}\right)\left[T_1(t-1) - T_2(t-1)\right]\right\}$$

where

$T_1(t)$ = temperature of the atmosphere and upper oceans

$T_2(t)$ = temperature of the deep oceans

R_1 = thermal capacity of the atmosphere and upper oceans

R_2 = thermal capacity of the deep oceans

$F(t)$ = radiative forcing in the atmosphere from green house gases

λ = the climate feedback parameter

$1/\tau_2$ = the transfer rate from the upper layer to the lower layer

This function appears complicated at first; however, taking it piece by piece makes it easier to understand. Consider first a simpler version of Eq. (5) with only the lagged T_1 term and the F term, i.e.

$$(6) \qquad T_1(t) = T_1(t-1) + \left(\frac{1}{R_1}\right)\left\{F(t) - \lambda T_1(t-1)\right\}$$

This is just a dynamic equation of the temperature of the atmosphere driven by the forcing term and mitigated by the climate feedback parameter λ. The other term in Eq. (5) is the difference between the atmosphere temperature T_1 and the deep oceans temperature T_2, i.e.

$$(7) \qquad -\left(\frac{R_2}{\tau_2}\right)\left[T_1(t-1) - T_2(t-1)\right]$$

Thus, because of the negative sign in front of the term in Eq. (7), the greater the difference between the two temperatures the less the atmosphere temperature will increase from one period to the next. So if the deep ocean is much cooler than the

atmosphere it will absorb heat and result in less increase in the atmosphere temperature. This can also be seen in the equation for the temperature of the deep oceans, i.e.

$$T_2(t) = T_2(t-1) + \left(\frac{1}{R_2}\right)\left\{\left(\frac{R_2}{\tau_2}\right)\left[T_1(t-1) - T_2(t-1)\right]\right\} \tag{8}$$

In this case there is a positive effect of the temperature difference between the two layers. Thus an increase in the difference between the atmosphere and deep oceans temperatures in period $t-1$ results in a more rapid increase in the deep ocean temperature in period t.

Next we need to close the loop of causation in the model from temperature back to output. First, recall the use of the Ω term in the production function in Eq. (1), i.e.

$$Q(t) = \Omega(t)\, A(t) K(t)^{\gamma} L(t)^{1-\gamma} \tag{1}$$

The Ω term in the Nordhaus model is driven by the d variable which is defined as

$$d(t) = a_1 \left[T_1(t) \Big/ 3 \right]^2 \tag{9}$$

where

$d(t)$ = fractional loss of global output from greenhouse warming

a_1 = a constant

Thus, as the temperature of the atmosphere rises, the fractional loss of global output increases in a nonlinear way.

The d term, in turn, appears in the denominator of the Ω term as follows

$$\Omega(t) = \frac{1 - TC(t)}{1 + d(t)} \tag{10}$$

where

$TC(t)$ = fractional cost to global output from green house gas emission controls

So as temperature increases the d term increases and thus the Ω term decreases and output declines. Also, the definition of the Ω term includes the term TC which represents the efficiency loss in output that is caused by the use of the carbon taxes. This loss is represented in the model with the equation

(11) $$TC(t) = b_1 \mu(t)^{b_2}$$

Thus as the carbon tax increases and μ, the fractional reduction of emissions, increases the efficiency loss term TC increases. Also, from Eq. (10), as this loss increases the Ω term decreases.

So in summary, the Ω term is indirectly affected by two variables, μ and T – both of which cause it to fall as they increase. The first variables is the fractional reduction of emissions, μ, operating through the TC variable and the second is temperature, T, operating through the d variable. However, the μ and T variables are related in an inverse fashion to one another in the model. As the carbon tax underlying μ increases the temperature T declines. This is the essential tradeoff in the model – higher carbon taxes reduce emissions, decrease temperature and increase output; however, they also impose efficiency loss on the economy and thus reduce output.

There is also a second basic tradeoff at work in the model. This comes from the fact that this model is basically a one-sector growth model of the Ramsey type that was modeled in Excel earlier in this book. The tradeoff in the growth model is between consumption and investment and is embodied in the equations

(12) $$Q(t) = C(t) + I(t)$$

(13) $$K(t) = (1 - \delta_K) K(t-1) + I(t)$$

where

$C(t)$ = total consumption in period t

$I(t)$ = investment in period t

δ_K = rate of depreciation of the capital stock

Thus as consumption rises investment must fall and as investment falls capital accumulation declines and thus output declines.

This is in turn linked to the criterion function of the model which is to maximize discounted utility

$$\max_{[c(t)]} \sum_{t=1}^{T} U\left[c(t), P(t)\right](1+\rho)^{-t} \tag{14}$$

where

$U[\ ]$ = utility function

$P(t)$ = population in period t

ρ = pure rate of social time preference

$c(t)$ = per capita consumption in period t

Also, the utility in each period is a nonlinear function of per capita consumption (actually, in this model, it is consumption per member of the labor force)

$$c(t) = C(t) / L(t) \tag{15}$$

where the utility function is the same general form as was used in the growth model in Excel, i.e.

$$U\left[c(t), L(t)\right] = L(t)\left\{\left[c(t)\right]^{1-\alpha} - 1\right\} / (1-\alpha) \tag{16}$$

In summary, this tradeoff is that as total consumption increases it increases per capita consumption and thus utility; however this is achieved by reducing investment and thus capital accumulation and thus reduces future output.

This completes the statement of the model. However, since the model is somewhat long it is useful to restate it in a summary fashion.

2. The Model in Summary

We begin with the criterion function and continue with the constraints

Criterion Function (from Eq. 14)

$$\max_{[c(t)]} \sum_{t=1}^{T} U\left[c(t), L(t)\right](1+\rho)^{-t} \tag{17}$$

Utility Function (from Eq. 16)

(18) $$U\left[c(t),L(t)\right]=L(t)\left\{\left[c(t)\right]^{1-\alpha}-1\right\}/(1-\alpha)$$

Production Function (from Eq. 1)

(19) $$Q(t)=\Omega(t)\,A(t)\,K(t)^{\gamma}\,L(t)^{1-\gamma}$$

Output Division (from Eq. 12)

(20) $$Q(t)=C(t)+I(t)$$

Per Capita Consumption (from Eq. 15)

(21) $$c(t)=C(t)/L(t)$$

Capital Accumulation (from Eq. 13)

(22) $$K(t)=(1-\delta_K)K(t-1)+I(t)$$

Emissions (from Eq. 2)

(23) $$E(t)=\left[1-\mu(t)\right]\sigma(t)Q(t)$$

CO_2 Concentration (from Eq. 3)

(24) $$M(t)=\beta E(t)+(1-\delta_M)M(t-1)$$

Temperature in the Atmosphere and Upper Oceans (from Eq. 5)

(25) $$T_1(t)=T_1(t-1)+\left(\frac{1}{R_1}\right)\left\{F(t)-\lambda T_1(t-1)-\left(\frac{R_2}{\tau_2}\right)\left[T_1(t-1)-T_2(t-1)\right]\right\}$$

Temperature in the Deep Oceans (from Eq. 8)

(26) $$T_2(t)=T_2(t-1)+\left(\frac{1}{R_2}\right)\left\{\left(\frac{R_2}{\tau_2}\right)\left[T_1(t-1)-T_2(t-1)\right]\right\}$$

Forcing Term (from Eq. 4)

(27) $$F(t)=4.1\left[\frac{\log M(t)}{\frac{590}{\log 2}}\right]+FO(t)$$

Fractional Loss of Output From Greenhouse Warming (from Eq. 9)

(28) $$d(t) = a_1 \left[T_1(t) \Big/ 3 \right]^2$$

Fractional Cost to Output from Controls – Carbon Taxes (from Eq. 11)

(29) $$TC(t) = b_1 \mu(t)^{b_2}$$

Climate and Emission Control Impact (from Eq. 10)

(30) $$\Omega(t) = \frac{1 - TC(t)}{1 + d(t)}$$

3. The Model in GAMS

The GAMS representation of Nordhaus's DICE model is in the file `dice.gms` and is listed in Appendix 15A. This implementation of the model uses 40 time periods each of which are ten years long, thus the model covers a time horizon of 400 years. It is not uncommon in dynamic models to have more than one year per time period; however, it does require some adjustments. For example, the capital accumulation equations is changed from

$$K(t) = (1 - \delta_K) K(t-1) + I(t)$$

to

$$K(t) = (1 - \delta_K)^{10} K(t-1) + 10\, I(t)$$

Since the depreciation rate is annual it is necessary to raise it to a power that is equal to the number of years per time period. Also, the flow variables, like investment in this equation, are in annual terms and must be multiplied by the number of years per time period in order to use them appropriately in accumulation equations.

Also, some of the other equations in the GAMS statement of the model are in a slightly different form than in the mathematics used above. In particular, the production function and the emission equations used in the GAMS statement are obtained by substitution of some equations.

The production function is created by substituting Eqs. (28) and (29) into Eq. (30) to obtain

$$\Omega(t) = \frac{1 - b_1 \mu(t)^{b_2}}{1 + a_1 \left[T_1(t) / 3 \right]^2} \tag{31}$$

or

$$\Omega(t) = \frac{1 - b_1 \mu(t)^{b_2}}{1 + \left(a_1 / 9 \right) T_1(t)^2} \tag{32}$$

And Eq. (32) is substituted into Eq. (19) i.e.

$$Q(t) = \Omega(t) A(t) K(t)^{\gamma} L(t)^{1-\gamma} \tag{19}$$

to obtain

$$Q(t) = \frac{1 - b_1 \mu(t)^{b_2}}{1 + \left(a_1 / 9 \right) T_1(t)^2} A(t) K(t)^{\gamma} L(t)^{1-\gamma} \tag{33}$$

and then rearranged to obtain

$$Q(t) = A(t) L(t)^{1-\gamma} K(t)^{\gamma} \left[\frac{1 - b_1 \mu(t)^{b_2}}{1 + \left(a_1 / 9 \right) T_1(t)^2} \right] \tag{34}$$

which is the form of the production function used in the GAMS statement.

Also, the emissions equation used in the GAMS representation is obtained by using Eq. (23), i.e.

$$E(t) = \left[1 - \mu(t) \right] \sigma(t) Q(t) \tag{35}$$

and substituting the production function from Eq. (19) into it to obtain

$$E(t) = \left[1 - \mu(t) \right] \sigma(t) \Omega(t) A(t) K(t)^{\gamma} L(t)^{1-\gamma} \tag{36}$$

Then Eq. (36) is rearranged to obtain

(37) $$E(t) = \sigma(t)\left[1-\mu(t)\right]\Omega(t)\,A(t)\,L(t)^{1-\gamma}\,K(t)^{\gamma}$$

and Ω to set to one to obtain

(38) $$E(t) = \sigma(t)\left[1-\mu(t)\right]A(t)\,L(t)^{1-\gamma}\,K(t)^{\gamma}$$

This last step of setting Ω to one is surprising so the user may want to restore a non-unitary Ω to that equation in the GAMS representation.

The parameter σ is treated as time varying in the equation

(39) $$\sigma(t) = \sigma_0 e^{g_\sigma(t)}$$

with

(40) $$g_\sigma(t) = \left(g_{\sigma 0} \big/ \delta_a\right)\left(1-e^{-\delta_a t}\right)$$

where

σ_0 = initial CO_2-equivalent emission-GNP ratio
$g_\sigma(t)$ = cumulative improvement of energy efficiency
$g_{\sigma 0}$ = growth of σ per decade
δ_a = decline rate of technological change per decade

The total factor productivity parameter in the production function is treated in a similar fashion with the equations

(41) $$A(t) = A_0 e^{g_a(t)}$$

with

(42) $$g_a(t) = \left(g_{a0} \big/ \delta_a\right)\left(1-e^{-\delta_a t}\right)$$

where

A_0 = initial level of total factor productivity
$g_a(t)$ = growth rate of productivity from 0 to T
g_{a0} = initial growth rate of technology per decade

Also the rate of growth of the labor force is treated in the same way with the equations

(43) $$L(t) = L_0 e^{g_L(t)}$$

with

(44) $$g_L(t) = \left(g_{L0} \Big/ \delta_L \right)\left(1 - e^{-\delta_L t}\right)$$

where

L_0 = 1965 world population in millions

$g_L(t)$ = growth rate of labor from 0 to t

g_{L0} = growth rate of population per decade

Finally the exogenous forcing term for other greenhouse gases is set using the equations

(45) $$FO(t) = 0.2604 + 0.125t - 0.0034t^2 \quad for\ t < 15$$
$$FO(t) = 1.42 \quad for\ t \geq 15$$

Thus this term increases in a quadratic way from 0.2604 to 1.42 over the first fifteen years and then remains constant at 1.42.

If you have already read the previous chapters in this book dealing with models in GAMS, particularly the dynamic models, the GAMS representation of the Nordhaus model provided in Appendix15A will seem familiar terrain. If you have not read the previous chapters with GAMS models, you are encouraged do so and to take a look at Appendix H on Stacking Method in GAMS and Appendix F on Introduction to Nonlinear Optimization Solvers.

4. Results

In the 1992 *Science* article Nordhaus compares five solutions of the model, which are shown in Table 15.1. The first solution is a "no-controls" result in which μ is set

Case	Policy	Base Value	Dollar Difference	Percent Difference
1	No-controls	731.694	0	0.000
2	Optimal Policy	731.893	199	0.027
3	Stabilize emissions	726.531	-5163	-0.706
4	Stabilize climate	701.764	-29930	-4.091
5	Geoengineering	735.787	4093	0.559

Table 15.1 Solutions of the Model

to zero, i.e. there is no removal of emissions relative to the uncontrolled level. The second solution is the full optimal control solution that provides a slight (0.027%) improvement in total discounted utility over the horizon covered by the model.

The third solution is to fix emissions at around 10% above the uncontrolled level after 1995. This requires setting μ equal to 0.1 after 1995 and can be implemented in the GAMS statement of the model by using a MIU.FX statement before the SOLVE statement. As is seen in Table 1 this results in a decrease in total discounted utility by about seven-tenths of a percent.

A more drastic policy is to stabilize climate as is shown in the fourth solution. This solution limits the temperature increase to $0.2^{\circ}C$ per decade after 1985 with an upper limit of a total increase of $1.5^{\circ}C$ from 1990. This results in a decrease of about 4 percent in total discounted utility relative to the uncontrolled solution.

The final solution considers the effects on introducing a hypothetical technology that provides costless mitigation of climate change. Examples cited by Nordhaus include shooting smart mirrors into space or seeding the ocean with iron to accelerate carbon sequestration.

5. Experiments

The obvious experiments with this model are to attempt to replicate some of the solutions shown in Table 1; however, there are a number of other experiments of interest. One such experiment is to decrease the size of parameter a_1 in Eq. (28).

(28) $$d(t) = a_1 \left[T_1(t) \Big/ 3 \right]^2$$

This experiment recognizes that there is considerable controversy about the magnitude of the effect of increases in temperature on economic output. In fact, some Russians seem to have concluded that because of the northerly location of most of their country that slight temperature increases might actually result in increases rather than in decreases of national GDP.

Another experiment would be to increase the parameter δ_M in the CO_2 concentration equation

(24) $$M(t) = \beta E(t) + (1 - \delta_M) M(t-1)$$

to reflect a feeling that the atmosphere is able to breakdown more of the CO_2 than the original parameter value reflects.

6. Further Reading

As was mentioned above, this chapter is based on the article by Nordhaus in *Science* in 1992 about the DICE model. That model has the virtue of being relatively simple and is thus useful for this chapter. For a later model see the RICE model by Nordhaus and Boyer (2000). For a model that is used to analyze the costs of CO2 emissions limits see Manne and Richels (1992). For an alternative to the IPCC CO2 emission projections see Eckaus (1994). For a general equilibrium model approach to the analysis of reducing carbon emissions see Blitzer, Eckaus, Lahiri and Meeraus (1992).

For a model that uses the GAMS software and focuses on the role of the developing countries in global warming - particularly India and China - see Duraiappah (1993). For particular reference to the effects of greenhouse gases in agriculture and

forestry see McCarl and Schneider (2001). For a discussion of climate policy change after the Kyoto treaty see McKibbin and Wilcoxen (2002).

For an example of the analysis of water pollution control with a GAMS model see Letson (1992). Those interested in environmental models for various sectors of the economy can finds models of the plastics sector in China, the pulp and paper sector in India, the shrimp industry in Thailand and the livestock sector in Botswana in Duraiappah (2003).

Appendix 15A

The GAMS Representation of the Global Warming Model

```
$offsymxref offsymlist
* Explaining the DICE, Cowles Foundation Discussion Paper, January 1991
* The calibration is to a 60-period run for the transversality
*
sets    t               time periods     /1*40/
        tfirst(t)       first period
        tlast(t)        last period

scalars bet     elasticity of marginal utility          /0/
        r       rate of social time preference per year /0.03/
        gl0     growth rate of population per decade    /0.223/
        dlab    decline rate of population growth per decade    /0.195/
        deltam  removal rate carbon per decade  /0.0833/
        ga0     initial growth rate for technology per decade   /0.15/
        dela    decline rate of technological change per year   /0.11/
        sig0    co2-equivalent emissions-gnp ratio      /0.519/
        gsigma  growth of sigma per decade      / - 0.1168 /
        dk      depreciation rate on capital per year /0.10/
        gama    capital elasticity in production function       /0.25/
        m0      co2-equivalent concentrations 1965 billions t c /677/
        tl0     lower stratum temperature (c) 1965      /0.10/
        t0      atmospheric temperature (c) 1965        /0.2/
        atret   marginal atmosphere retension rate      /0.64/
        q0      1965 world gross output trillion 89 US$    /8.519/
        ll0     1965 world population million   /3369/
        k0      1965 value capital trillion 1989 US$    /16.03/
        c1      climate-equation coefficient for upper level    /0.226/
        lam     climate feedback factor /1.41/
        c3      transfer coefficient upper to lower stratum     /0.440/
        c4      transfer coefficient for lower level    /0.02/
        a0      initial level of total factor productivity   /0.00963/
        a1      damage coeff for co2 doubling(fraction GWP)  /0.0133/
        b1      intercept control cost function /0.0686/
        b2      exponent of control cost function       /2.887/
        phik    transversality coeff capital ($ per unit)        /140/
        phim    transversality coeff carbon ($ per unit)  / - 9.0 /
        phite   transversality coeff temperature ($ per unit)/ - 7000 /
```

```
parameters      l(t)    level of population and labour
                al(t)   level of total factor productivity
                sigma(t)        co2-equvalent-emissions output ratio
                rr(t)   discount factor
                ga(t)   growth rate of productivity from 0 to t
                forcoth(t) exogenous forcing for other greenhouse gases
                gl(t)   growth rate of labour 0 to t
                gsig(t) cumulative improvement of energy-efficiency
                dum(t)  dummy variable 0 except last period ;

        tfirst(t) = yes$(ord(t) eq 1);
        tlast(t)  = yes$(ord(t) eq card(t));
        display tfirst, tlast;

        gl(t) = (gl0/dlab)*(1-exp(-dlab*(ord(t)-1)));
        l(t)  = ll0*exp(gl(t));
        ga(t) = (ga0/dela)*(1-exp(-dela*(ord(t)-1)));

        al(t) = a0*exp(ga(t));
        gsig(t) = (gsigma/dela)*(1-exp(-dela*(ord(t)-1)));
        sigma(t) = sig0*exp(gsig(t));

        dum(t) = 1$(ord(t) eq card(t));
        rr(t)  = (1+r)**(10*(1-ord(t)));

        forcoth(t)  =  1.42;
        forcoth(t)$(ord(t) lt 15) = 0.2604 + 0.125*ord(t)
                                    - 0.0034*ord(t)**2;

variables       miu(t)  emission control rate GHGs
                forc(t) radiative forcing, W per m2
                te(t)   temperature, atmosphere C
                tl(t)   temperature, lower ocean C
                m(t)    co2 equivalent concentration bill t
                e(t)    co2 equivalent emissions bill t
                c(t)    consumption trillion US$
                k(t)    capital stock trillion US$
                cpc(t)  per-capita consumption 1000s US$
                pcy(t)  per-capita income 1000s US$
                i(t)    investment trillion US$
                s(t)    savings rate as fraction of GWP
                ri(t)   real interest rate per annum
                trans(t)        transversality variable last period
                y(t)    output

                utility;
```

```
positive variables miu, e, te, m, y, c, k, i;

equations       util    objective function
                yy(t)   output equation
                cc(t)   consumption equation
                kk(t)   capital balance equation
                kk0(t)  initial condition for k
                kc(t)   terminal condition for k
                cpce(t) per-capita consumption definition
                pcye(t) per-capita income definition
                ee(t)   emissions process
                seq(t)  savings rate equation
                rieq(t) interest rate equation
                force(t)  radiative forcing equation
                mm(t)   co2 distribution equation
                mm0(t)  initial condition for m
                tte(t)  temperature-climate equation for atmosphere
                tte0(t)  initial condition for atmospheric temperature
                tle(t)  temperature-climate equation for lower oceans
                transe(t) transversality condition
                tle0(t) initial condition for lower ocean ;

* Equations of the model

kk(t)..         k(t+1) =l= (1-dk)**10*k(t) + 10*i(t) ;
kk0(tfirst)..   k(tfirst) =e= k0 ;
kc(tlast)..     r*k(tlast) =l= i(tlast) ;

ee(t)..         e(t) =g= 10*sigma(t)*
                     (1 - miu(t))*al(t)*l(t)**(1 - gama)*k(t)**gama ;
force(t)..      forc(t) =e= 4.1*(log(m(t)/590)/log(2)) + forcoth(t) ;
mm0(tfirst)..   m(tfirst) =e= m0 ;
mm(t+1)..       m(t+1) =e= 590 + atret*e(t) + (1-deltam)*(m(t) - 590) ;

tte0(tfirst)..  te(tfirst) =e= t0 ;
tte(t+1)..      te(t+1) =e= te(t)+c1*(forc(t)-lam*te(t)
                            - c3*(te(t)-tl(t))) ;
tle0(tfirst)..  tl(tfirst) =e= tl0 ;
tle(t+1)..      tl(t+1) =e= tl(t) + c4*(te(t) - tl(t));

yy(t)..         y(t) =e= al(t)*l(t)**(1-gama)*k(t)**gama
                         *(1-b1*(miu(t)**b2))/(1+(a1/9)*sqr(te(t)));
seq(t)..        s(t) =e= i(t)/(.001+y(t)) ;
rieq(t)..       ri(t) =e= gama*y(t)/k(t) - (1-(1-dk)**10)/10 ;
```

```
cc(t)..          c(t) =e= y(t) - i(t) ;
cpce(t)..        cpc(t) =e= c(t)*1000/l(t) ;
pcye(t)..        pcy(t) =e= y(t)*1000/l(t) ;

transe(tlast).. trans(tlast) =e= rr(tlast) *(phik*k(tlast)
                                 + phim *m(tlast)+phite*te(tlast));

util..           utility =e= sum(t,10*rr(t)*l(t)*log(c(t)/l(t))
                            /0.55+trans(t)*dum(t));

* Upper and lower bounds; general conditions imposed for stability

miu.up(t) = 0.99;
miu.lo(t) = 0.01;
k.lo(t) = 1;
te.up(t) = 20;
m.lo(t) = 600 ;
c.lo(t) = 2;

* Upper and lower bounds for historical constraints

miu.fx('1') = 0.0;
miu.fx('2') = 0.0;
miu.fx('3') = 0.0;

* Solution options

option iterlim = 99999;
option reslim = 99999;
option solprint = off;
option limrow = 0;
option limcol = 0;

model co2 /all/ ;
solve co2 maximising utility using nlp ;

* Display of results

display y.l, c.l, s.l, k.l, miu.l, e.l, m.l, te.l, forc.l, ri.l ;
display cc.m, ee.m, kk.m, mm.m, tte.m, cpc.l, tl.l, pcy.l, i.l ;
display sigma, rr, l, al, dum, forcoth ;
```

Chapter 16

Dynamic Optimization in MATLAB

Dynamic optimization encompasses a group of mathematical techniques used in economics to model the intertemporal behavior of economic agents under the assumption of forward looking optimizing behavior. For example, it can be used to model the behavior of a policymaker who tries to determine the optimal path of policy variables in order to achieve some specified targets for GDP or the inflation rate; to model the behavior of firms that are assumed to choose the optimal path of investment in order to maximize intertemporal profits or their present value; to model the behavior of consumers who are assumed to face intertemporal choices between present and future consumption; etc.

In general terms, dynamic optimization deals with the problem of obtaining a sequence of optimal choices under given dynamic constraints. Calculus of variations, optimal control and dynamic programming are the most commonly used techniques for dynamic optimization. In this chapter, we will focus on what is known as discrete time dynamic programming, a technique particularly suited for computational implementation given its recursive structure. Specifically, we will deal with a special case of dynamic optimization that is known as the Quadratic Linear Problem (QLP), a very popular kind of problem in which the goal is to optimize an intertemporal quadratic objective function subject to dynamic linear constraints that hold as equalities.[35] [36] The QLP is used here for a deterministic model because it is also well adapted and widely used for the types of stochastic models that we will progress to. We have already dealt with a QLP earlier in the book in the chapter on Thrift in GAMS. However, in that chapter we did not exploit the recursive nature of the typical QLP. There, we solved the problem with nonlinear

[35] Though the choice of the quadratic criterion can be somewhat limiting many nonlinear models can be usefully approximated by QLP models and then solved with successive approximations.

[36] For dynamic models in which there are inequalities, mathematical programming methods like those used with GAMS in the chapter on global warming are more appropriate than the Riccati methods discussed in this chapter. On the other hand the quadratic linear control theory models with equality constraints are most useful when one wants to deal with stochastic elements in the form of additive noise terms and uncertain parameters.

programming in GAMS. That approach can easily deal with inequality constraints; however, it does not use a recursive solution method. Rather it uses a "stacking" method which transforms a dynamic problem into a larger static one.

In this chapter we turn to MATLAB which uses a vector-matrix paradigm more suitable to deal with the standard QLP since, as we will see below, the solution of these problems involves a series of vector and matrix operations. We have already introduced MATLAB in earlier chapters. In Appendix 16A we provide the listing and in the book web site we provide the file for the MATLAB representation of the model we will develop in this chapter. This code was based on an earlier code in GAUSS by Hans Amman and was created in MATLAB by Huber Salas and Miwa Hattori.

This chapter begins with a brief introduction to the mathematics of QLP. Then, as a simple example, a small macroeconometric model is introduced. Finally, the model is input to MATLAB and solved.

1. Introduction to Dynamic Programming

The dynamic programming approach, developed by Richard Bellman (1957), can be illustrated with a simple example. The diagram in Figure 16.1 represents different ways of going from node x_{11} to x_{41} applying a specific action u to move from one node to another. As a concrete example, we can interpret the nodes as towns and the actions as means of transportation (car, plane, train, etc.). Each town has associated a cost (e.g. room and board). Also each means of transportation has a cost associated to it. The problem is to find the minimum cost path or, more precisely in the case of dynamic programming, a feedback rule to determine the optimal action u as a function of the node we are at.

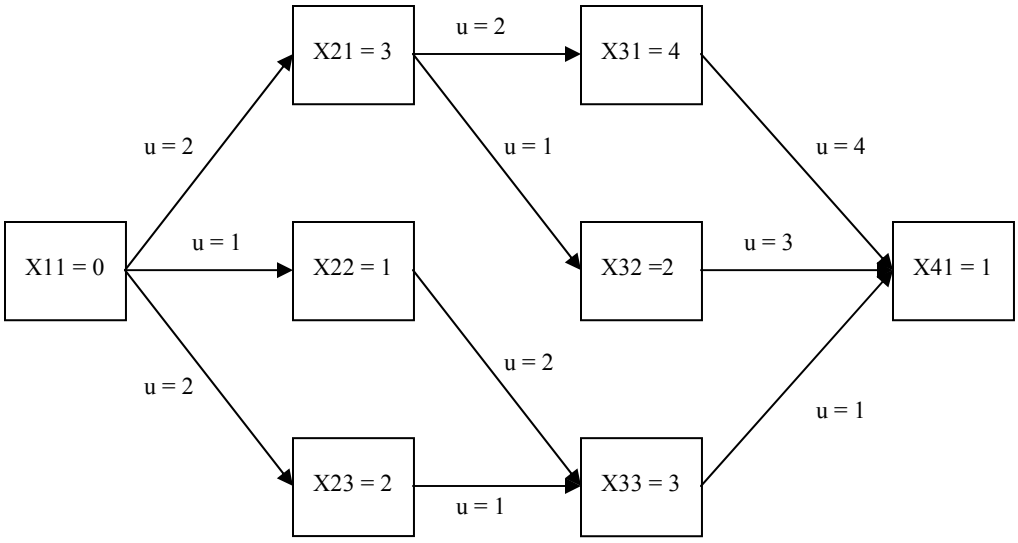


Figure 16.1 Dynamic Programming Example

In a more general formulation, we can think of the diagram in Figure 16.1 as representing the time path of a system that can be driven from one state (node) to another by manipulating controls (the *u*'s). Going back to a more concrete example, now in time and not in space as above, you can think of a macroeconomic example in which the state variable is the inflation rate, or alternatively the level of GDP, and the control variable is the money supply. The problem would be the one faced by the monetary authority trying, at a minimum cost for society, to drive inflation down, or the GDP level up, facing a number of alternative economic paths to achieve that goal.

To solve the problem, the dynamic programming approach uses a recursive method that works backwards. For the example at hand, the method works as follows.

1) Compute the cost *J* of each segment in the last stage (that is, add the cost of x_{41} and the cost of the corresponding *u*). There are obviously three values:

$$J(x_{31}, x_{41}) = 5; \quad J(x_{32}, x_{41}) = 4; \quad J(x_{33}, x_{41}) = 2$$

2) Compute the cost of each feasible optimal sequence of segments from the $x_{2..}$ nodes:

$$J(x_{21},x_{41}) = \min\{\, J(x_{31},x_{41}) + \text{cost of segment } (x_{21},x_{31});$$
$$J(x_{32},x_{41}) + \text{cost of segment } (x_{21},x_{32})\,\}$$
$$= \min\{5+6;\ 4+3\} = 7$$

which implies that the optimal sequence of controls to go from x_{21} to x_{41} is $[\,u(x_{21},x_{32}),\ u(x_{32},x_{41})\,]$. Obviously, the optimal sequence (x_{22},x_{41}) is $[\,u(x_{22},x_{33}),u(x_{33},x_{41})\,]$ (the only feasible) with $J(x_{22},x_{41}) = 7$, while for (x_{23},x_{41}) is $[\,u(x_{23},x_{33})\ u(x_{33},x_{41})\,]$ with $J(x_{23},x_{41}) = 6$.

3) Compute the cost of the optimal feasible sequence of segments from x_{11}:

$$J(x_{11},x_{41}) = \min\{\, J(x_{21},x_{41}) + \text{cost of segment } (x_{11},x_{21});$$
$$J(x_{22},x_{41}) + \text{cost of segment } (x_{11},x_{22});$$
$$J(x_{23},x_{41}) + \text{cost of segment } (x_{11},x_{23})\,\}$$
$$= \min\{7+5;\ 7+2;\ 6+4\} = 9$$

Thus the optimal sequence of controls for the problem is

$$[\,u(x_{11},x_{22}),u(x_{22},x_{33}),u(x_{33},x_{41})\,].$$

2. A Simple Quadratic Linear Problem

In most economic applications we find problems in which we represent an economic agent, institution or the economy as a whole as a system of state variables that evolves through time. This system can be manipulated by means of a set of control variables in order to minimize (or maximize) an intertemporal cost (or value) function. A very typical problem is one in which the cost function is quadratic and the economic system is represented by linear equations. For a very simple one-state one-control case, the problem is expressed as one of finding the controls $\left(u_k\right)_{k=0}^{N-1}$ to minimize a quadratic criterion function J of the form:

$$(1) \qquad J = \frac{1}{2}\sum_{k=0}^{N} x_k^2$$

subject to the dynamic equation:

(2) $x_{k+1} = ax_k + bu_k$

and the initial condition

(3) x_0

where:

x = state variable
u = control variable
a = state parameter
b = control parameter

As we already know, the dynamic programming approach works by solving the problem backward in time, determining optimal feedback rules for choosing the control vector as a function of the state vector at each stage - each time period - of the problem. Thus it transforms the original optimization problem into a sequence of sub-problems. Its crucial notion is the optimal cost-to-go, which is the cost along the minimum-cost path from a given time period to the terminal period of the problem. For QLP, the cost-to-go is a quadratic function of the state of the system at time *k*, which for our particular problem is

(4) $$J^*(x_k) = \frac{1}{2}x_k^2$$

Starting from the terminal period, the cost is

(5) $$J^*(x_N) = \frac{1}{2}x_N^2 .$$

The optimal cost-to-go at period *N-1* will be the minimum of the optimal cost-to-go at state x_N in time *N* and the cost incurred in time period *N-1*

(8) $$J^*(N-1) = \min_{u_{N-1}}\left\{J^*(N) + L_{N-1}\left(x_{N-1}\right)\right\}$$

where L_{N-1} is the cost function *J* for *N-1* in Eq. (1). Thus we have

$$J^*(N-1) = \min_{u_{N-1}} \left\{ \frac{1}{2} x_N^2 + \frac{1}{2} x_{N-1}^2 \right\} \tag{9}$$

To carry on the minimization we need, in Eq. (9), all variables expressed at time *N-1*. Substituting Eq. (2) for x_N into (9) and expanding we obtain

$$J^*(N-1) = \min_{u_{N-1}} \left\{ \frac{1}{2} x_{N-1}^2 a^2 + x_{N-1} u_{N-1} ab + \frac{1}{2} u_{N-1}^2 b^2 + \frac{1}{2} x_{N-1}^2 \right\} \tag{10}$$

The first order condition for the minimization is

$$\frac{\partial \left\{ J^*(N-1) \right\}}{\partial u_{N-1}} = x_{N-1} ab + u_{N-1} b^2 = 0 \tag{11}$$

Solving (11) for u_{N-1} we obtain a feedback rule that gives us the optimal control as a function of the state

$$u_{N-1} = G_{N-1} x_{N-1} \tag{12}$$

where G_{N-1}, known as the feedback gain coefficient, is

$$G_{N-1} = -\frac{a}{b} \tag{13}$$

If we repeat the procedure for $J^*(N-2)$, etc., we will observe that a general form for the feedback rule emerges

$$u_k = G_k x_k = -\frac{a}{b} x_k \tag{14}$$

Thus, the feedback rule (14) tells us what the optimal action to take is at each point in time depending of the state of the system. For our particular problem the feedback gain coefficient is a constant. However, as we will se later, this is not the case

for more general problems.

To obtain the solution paths for the controls and the states, we have to start from the initial condition (3) to obtain the optimal control from Eq. (14). Then we can solve Eq. (2) to obtain the next optimal state and go back to Eq. (14) to obtain the optimal control and so on. Thus we can see that the solution paths are obtained from a "forward loop". Knowing the optimal states, we can compute the corresponding criterion value from Eq. (1).

The MATLAB representation of the solution procedure of the simple problem we presented is straightforward, and it is available in the book web site in file `qlpsimple.m`. We begin by initializing the problem for four periods (from zero to three), and we assign values for the parameters and the initial condition. We also set to zero what will be the vectors containing the optimal states and controls, the variable `sum` which will contain the criterion value, and the index variable `k`.

```
t = 3;   a = 0.7;   b = -0.3;   x0 = -1;
u = zeros(1,t);   x = zeros(1,t);
sum = 0; k = 0;
```

Next we write the forward loop.

```
xold = x0;
while k <= t;
  glarge = - a / b;
  uopt = glarge * xold;
  xnew = a * xold + b * uopt;
  sum = sum + 0.5 * xold^2;
  x(1,k+1) = xold;
  u(1,k+1) = uopt;
  xold = xnew;
  k = k+1;

end;
```

The loop begins with the assigment of the initial condition to the `xold` variable, and it will run as long as `k <= t`. At each pass, the values of the feedback gain coefficient `glarge`, the optimal control `uopt` and the optimal state `xnew` are computed and stored in the corresponding positions of vectors `x` and `u`, and the corresponding value of the criterion function is computed and added to the `sum` variable. Finally, the results are printed, previously transposing the vectors so that the results are displayed in colums.

```
u = u'
x = x'
Criterion = sum
```

The solution values for the optimal states and controls are shown in Table 16.1.

k	x	u
0	-1	-2.3333
1	0	0
2	0	0
3	0	0

Table 16.1 Optimal States and Controls

Notice that the value of `x` is driven to zero in just one period and with a single control action. Even if we extend the number of periods, and whatever initial condition we use, just one single control action will suffice. Why? The answer lies in the fact that the control variable is not a part of the criterion function to be minimized. Thus, there is no cost associated to the use of the control and this one can immediately jump to any necessary value to bring the state variable to zero. However, in most cases this variable will be included into the criterion function. Moreover, both the state variable and the control variable will have associated specific weights in the function to represent relative priorities in terms of the cost of having the state variable off-target versus the cost of using the control.

In the next section we will present a more comprehensive and general problem. It will be a many-state many-control problem. There will be weights on states and controls and also cross terms in the criterion function. And the system of equations will contain constant terms. We will see that the main logic to obtain a solution is the same we used in this section. However, some new elements will appear as a part of it such as Riccati matrices and vectors, and the solution procedure will involve a backward loop together with a forward loop similar to the one we presented in this section.

3. A More General Quadratic Linear Problem

The Quadratic Linear Problem (QLP) has linear system equations and a quadratic criterion and may be written as find

$$\left(u_k\right)_{k=0}^{N-1}$$

to minimize the criterion

$$(15) \qquad J=\frac{1}{2}x_N'W_Nx_N+w_N'x_N+\sum_{k=0}^{N-1}\left(\frac{1}{2}x_k'W_kx_k+w_k'x_k+x_k'F_ku_k+\frac{1}{2}u_k'\Lambda_ku_k+\lambda_k'u_k\right)$$

subject to the system equations

$$(16) \qquad x_{k+1}=Ax_k+Bu_k+c \qquad k=0,1,\cdots,N-1$$

and the initial conditions

$$(17) \qquad x_0 \quad \text{given}$$

where

$x_k = state\ \ vector$
$u_k = control\ \ vector$
$W_k = state\ vector\ \ priority\ matrix$
$F_k = cross\ state-control\ \ priority\ matrix$
$\Lambda_k = control\ vector\ \ priority\ matrix$
$A,\ B,\ and\ \ c = parameter\ \ matrices\ \ and\ \ vectors$

Also the notation

$$\left(u_k\right)_{k=0}^{N-1}$$

means the set of control vectors from period zero through period *N - 1*, that is $\left(u_0,u_1,u_2,\cdots,u_{N-1}\right)$. Period N is the terminal period of the model. Thus the problem is to find the time paths for the m control variables in each period for the time periods from *0* to *N - 1* to minimize the quadratic form (15) while starting at the initial conditions (17)

and following the difference equations (16). The derivation of the solution for this model is described in detail in Chapter 2 of Kendrick (1981). Here we will provide an outline of the procedure.

We know that for QLP, the cost-to-go is a quadratic function of the state of the system at time *k*, which for our problem is

$$J^*(x_k) = \frac{1}{2} x_k' K_k x_k + p_k' x_k + v_k \tag{18}$$

where K_k and p_k are called the Riccati matrix and vector respectively, and where v_k is a scalar. Starting from the terminal period we have

$$J^*(x_N) = \frac{1}{2} x_N' K_N x_N + p_N' x_N + v_N . \tag{19}$$

From Eq. (15), the cost at the terminal period is

$$\frac{1}{2} x_N' W_N x_N + w_N' x_N \tag{20}$$

Thus, from Eqs. (19) and (20) we obtain the result that $v_N = 0$ and the terminal conditions

$$K_N = W_N \tag{21}$$

$$p_N = w_N . \tag{22}$$

The optimal cost-to-go at period *N-1* will be the minimum of the optimal cost-to-go at state x_N in time *N* and the cost incurred in time period *N-1*

$$J^*(N-1) = \min_{u_{N-1}} \left\{ J^*(N) + L_{N-1}(x_{N-1}, u_{N-1}) \right\} \tag{23}$$

where L_{N-1} is the cost function *J* for *N-1* in Eq. (15). Analogously, the optimal cost-to-go at period *N-2* will be

$$J^*(N-2) = \min_{u_{N-2}} \left\{ J^*(N-1) + L_{N-2}(x_{N-2}, u_{N-2}) \right\} \tag{24}$$

and so on. Carrying out the corresponding substitutions in Eqs. (23) and (24), minimizing, solving the first order conditions with respect to the control vectors and rearranging, we will observe the emergence of a general solution which has the form of a feedback rule

(25) $$u_k = G_k x_k + g_k$$

where

(26) $$G_k = -\left[B'K_{k+1}B + \Lambda_k'\right]^{-1}\left[F_k + B'K_{k+1}A\right]$$

(27) $$g_k = -\left[B'K_{k+1}B + \Lambda_k'\right]^{-1}\left[B'\left(K_{k+1}c + p_{k+1}\right) + \lambda_k\right]$$

and where the expressions for the Riccati matrix and vector are

(28) $$K_k = A'K_{k+1}A + W_k - \left[A'K_{k+1}B + F\right]\left[B'K_{k+1}B + \Lambda_k'\right]^{-1}\left[F' + B'K_{k+1}A\right]$$

(29) $$p_k = -\left[A'K_{k+1}B + F\right]\left[B'K_{k+1}B + \Lambda_k'\right]^{-1}\left[B'\left(K_{k+1}c + p_{k+1}\right) + \lambda_k\right] + A'\left(K_{k+1}c + p_{k+1}\right) + w_k .$$

These equations look formidable but they essentially involve only the matrices and vector A, B and c from the system equations (16) and the matrices W and Λ from the criterion function (15) in addition to the lead values of the Riccati matrix K_{k+1} and the Riccati vector p_{k+1}.

To obtain the solution paths for the controls and the states, we have to start at the end and work backward. We will follow that procedure here by beginning with the terminal conditions and working back to the initial period while solving for the Riccati matrices and vectors. Then we will use the initial conditions, the feedback rule and the system equations to solve forward in time while computing the state and control variables.

Thus we begin by integrating backward in time starting with the terminal conditions

(30) $$K_N = W_N$$

(31) $$p_N = w_N$$

The backward integration is done by solving the Riccati matrix and vector equations

$$(32) \qquad K_k = A'K_{k+1}A + W_k - \left[A'K_{k+1}B + F\right]\left[B'K_{k+1}B + \Lambda_k'\right]^{-1}\left[F' + B'K_{k+1}A\right]$$

$$(33) \quad p_k = -\left[A'K_{k+1}B + F\right]\left[B'K_{k+1}B + \Lambda_k'\right]^{-1}\left[B'\left(K_{k+1}c + p_{k+1}\right) + \lambda_k\right] + A'\left(K_{k+1}c + p_{k+1}\right) + w_k$$

As will be shown later in the computational section of this chapter, these calculations can be programmed into MATLAB and solved in a very straightforward way. Using Eqs. (30) thru (33) the procedure is to begin with K_N which is obtained from Eq. (30). This matrix is then used first in Eq (32) to compute K_{N-1}, then K_{N-1} is used in that equation again to compute K_{N-2} etc until the K_0 matrix has been computed. A similar procedure is used to calculate the Riccati vectors p_k using Eqs. (31) and (33).

Once all the Riccati matrices and vectors have been computed, then a forward integration loop is started. In this loop the feedback gain matrix G_k and vector g_k from Eqs. (26) and (27) are computed for each time period using those expressions, i.e.

$$(34) \qquad G_k = -\left[B'K_{k+1}B + \Lambda_k'\right]^{-1}\left[F_k + B'K_{k+1}A\right]$$

and

$$(35) \qquad g_k = -\left[B'K_{k+1}B + \Lambda_k'\right]^{-1}\left[B'\left(K_{k+1}c + p_{k+1}\right) + \lambda_k\right]$$

These expressions - like those for the Riccati matrix and vector above are somewhat complicated but are easily programmed and solved in MATLAB.

The feedback gain matrix and vector are then used in the feedback rule

$$(36) \qquad u_k = G_k x_k + g_k$$

along with the initial condition, x_0, to compute the control vector, u_0. This value and the initial state, x_0, are then used in the system equations from Eq. (16) i.e.

$$(37) \qquad x_{k+1} = Ax_k + Bu_k + c$$

to compute the value of the state vector for the next period, x_1. Then the process is repeated beginning with Eqs. (34) and (35) and using Eqs. (36) and (37) until the control vectors, u_k, and the state vectors, x_k, have been computed for all time periods.

Also, in each pass through the forward loop the criterion value for that period is computed and added to the amount already accumulated using part of the criterion function, which is shown here in the tracking version rather than in the quadratic form, i.e.,[37]

$$\frac{1}{2}\left(x_k - \tilde{x}_k\right)' W_k \left(x_k - \tilde{x}_k\right) + \frac{1}{2}\left(u_k - \tilde{u}_k\right)' \Lambda_k \left(u_k - \tilde{u}_k\right) \tag{38}$$

where

$$\tilde{x}_k = desired \;\; state \;\; vector$$
$$\tilde{u}_k = desired \;\; control \;\; vector$$

We shall see shortly how intuitive it is to represent the mathematics of the solution procedure in MATLAB. Before doing so, we will comment on other types of problems and solution procedures that arise when we move from the deterministic framework of the standard QLP problem to an environment in which uncertainty is taken into account.

The simplest way of introducing uncertainty is by assuming that it takes the form of additive uncertainty. That is, assuming that the system of linear equations is shocked in each period by additive noise. When we do this, the mathematical representation of the QLP problem is modified in two ways. First, the objective function is now an expected value, thus Eq. (15) becomes:

(39)

$$E\{J\} = E\left\{\frac{1}{2}x_N' W_N x_N + w_N' x_N + \sum_{k=0}^{N-1}\left(\frac{1}{2}x_k' W_k x_k + w_k' x_k + x_k' F_k u_k + \frac{1}{2}u_k' \Lambda_k u_k + \lambda_k' u_k\right)\right\}$$

where *E* is the mathematical expectation operator. Second, the system of equations now has an additive noise term denoted by ξ_k, that is:

---

[37] Later in the chapter we discuss the transformation of the quadratic tracking version of the criterion function to the quadratic form.

(40) $$x_{k+1} = Ax_k + Bu_k + c + \xi_k$$

with

$$E\{\xi_k\} = 0 \qquad E\{\xi_k \xi_k'\} = Q_k \qquad E\left\{\xi_k \xi_{\substack{j \\ j \neq k}}'\right\} = 0$$

However, it can be shown that the solution procedure for this stochastic problem is the same as the one for the QLP problem (Eqs. (30) to (36)).[38] This is why the solution procedure when additive uncertainty is present is named Certainty Equivalent (CE). Notice that this does not mean that the observed optimal paths for the states and the controls will be the same in a QLP and a CE simulation, since in the CE simulation the system of equations will be shocked by additive noise at each time period.

Consider now the case of multiplicative uncertainty. In this case, we have information about the variances and covariances of the dynamic equations parameters (matrices *A*, *B* and vector *c*) and we want to exploit that knowledge when computing the optimal values of the controls to be applied period after period. In formal terms equations (39) and (40) still characterize the problem. However, it can be shown that the solution procedure is now somewhat different from the one corresponding to Eqs. (30) thru (36). Indeed, the expectations operator will appear now in those equations, as shown in Eqs. (43) - (46) below.

(41) $$K_N = W_N$$

(42) $$p_N = w_N$$

(43) $$K_k = E\{A'K_{k+1}A\} + W_k - \left[E\{A'K_{k+1}B\} + F\right]\left[E\{B'K_{k+1}B\} + \Lambda_k'\right]^{-1}\left[F' + E\{B'K_{k+1}A\}\right]$$

(44)

$$p_k = -\left[E\{A'K_{k+1}B\} + F\right]\left[E\{B'K_{k+1}B\} + \Lambda_k'\right]^{-1}\left[E\{B'K_{t+1}c\} + E\{B\}' p_{k+1} + \lambda_k\right] + E\{A'K_{k+1}c\} + E\{A\}' p_{k+1} + w_k$$

(45) $$G_k = -\left[E\{B'K_{k+1}B\} + \Lambda_k'\right]^{-1}\left[F_k + E\{B'K_{k+1}A\}\right]$$

(46) $$g_k = -\left[E\{B'K_{k+1}B\} + \Lambda_k'\right]^{-1}\left[E\{B'K_{k+1}c\} + E\{B\}' p_{k+1} + \lambda_k\right]$$

[38] See Kendrick (1981), Ch. 5.

Notice that there are now several terms involving the expectations of matrix products. To compute these expectations, following Chapter 6 in Kendrick (1981) we have to proceed as follows. In general terms, define:

(47) $D \equiv A'KB$

where *D, A, K, B* are all matrices, *A* and *B* are random and *K* is deterministic. Thus

(48) $E\{D\} = E\{A'KB\}$.

A single element in *D* is d_{ij}. Then:

(49) $E\{d_{ij}\} = E\{a_i'Kb_j\}$

where a_i is the *ith* column of *A* and b_j is the *jth* column of *B*. It can be shown that

(50) $E\{d_{ij}\} = (E\{a_i\})' KE\{b_j\} + tr[K\Sigma_{b_j a_i}]$

where

(51) $\Sigma_{b_j a_i} = E\left\{[b_j - E\{b_j\}][a_i - E\{a_i\}]'\right\}$

is the covariance matrix for the *jth* column of *B* and the *ith* column of *A* and $tr[\cdot]$ is the trace operator, i.e., the sum of the diagonal elements of the matrix in brackets.

To obtain the optimal paths for the controls and the states, we have to apply Eqs. (41) thru (46) in the same manner with backward and forward loops as we did for the deterministic QLP case presented earlier.

When a procedure like the one presented above is used to solve a problem in which multiplicative uncertainty is present, we say that we have an Open Loop Feedback without update problem (OLF w/o update). Why do we say "without update"? In more complex simulations like the ones to be presented in Section 4 of the next chapter, given the knowledge of the variances and covariances of the state equations parameters, we can consider a passive learning process. To do so, in each time period of the solution a projection-updating mechanism - usually a Kalman filter - is added to the solution

method of the optimization problem in order to obtain, in each period, updated values of next period parameters and of their variance-covariance matrix. You will have a better idea of this once you get to Section 4 of the next chapter. Details of the mathematical form of these procedures are in Kendrick (1981) (2002).

In the following we turn to the MATLAB representation of the QLP solution procedure. Before doing so, in the next section we will introduce a small macroeconomic model to use as an example. CE, OLF and parameter updating procedures and examples will be introduced in the next chapter using Duali, a high-level software especially designed to deal with these types of problems.

4. The Macroeconomic Model

The model is based on the work of Chow (1967) and Abel (1975) and is a very simple model with two state variables and two control variables that was used early in the control literature to perform some policy experiments. It was not chosen because of its nice properties but rather because it is very easy to implement in MATLAB and thus provides a good starting point to handle the more complex and realistic models like the ones to be presented in the next two chapters. The two state variables are

C_k = consumption
I_k = investment

and the control variables are

G_k = government expenditures
M_k = money supply

The reduced form of the model when estimated with data for the period 1954-II to 1963-IV as reported in Kendrick (1982a), is

$$C_{k+1} = 0.914C_k - 0.016I_k + 0.305G_k + 0.424M_k - 59.437 \tag{52}$$

$$I_{k+1} = 0.097C_k + .424I_k - 0.101G_k + 1.459M_k - 184.766 \tag{53}$$

Notice that the model exhibits "crowding out" behavior since the sign on the government expenditure variable in the investment equation is negative.

The model can be written in the notation of the system equations (16) above as

(16) $$x_{k+1} = Ax_k + Bu_k + c \qquad k = 0,1,\cdots,N-1$$

with

$$x_k = \begin{bmatrix} C_k \\ I_k \end{bmatrix} \qquad u_k = \begin{bmatrix} G_k \\ M_k \end{bmatrix}$$

$$A = \begin{bmatrix} 0.914 & -0.016 \\ 0.097 & 0.424 \end{bmatrix} \qquad B = \begin{bmatrix} 0.305 & 0.424 \\ -0.101 & 1.459 \end{bmatrix} \qquad c = \begin{bmatrix} -59.437 \\ -184.766 \end{bmatrix}$$

The initial conditions for the model are given by the values of consumption and investment respectively in 1964-I as

(54) $$x_0 = \begin{bmatrix} 387.9 \\ 85.3 \end{bmatrix}$$

The criterion for the model is of the form

(55) $$J = \frac{1}{2}\left(x_N - \tilde{x}_N\right)' \hat{W}_N \left(x_N - \tilde{x}_N\right) + \sum_{k=0}^{N-1}\left[\frac{1}{2}\left(x_k - \tilde{x}_k\right)' \hat{W}_k \left(x_k - \tilde{x}_k\right) + \frac{1}{2}\left(u_k - \tilde{u}_k\right)' \hat{\Lambda}_k \left(u_k - \tilde{u}_k\right)\right]$$

This is called a "tracking function" since it is minimized by having the optimal state and control vectors x_k and u_k track as closely as possible the desired state and control vectors $\tilde{x}_k$ and $\tilde{u}_k$. So the decision maker chooses the optimal time paths for the desired states and controls and then solves the model to compute the optimal controls which come as close as possible to these desired paths while satisfying the dynamic relationships in the system equations (16).

Compare Eq (55) to the criterion which was discussed in the mathematics section of this chapter, i.e. Eq. (15)

(15) $$J = \frac{1}{2}x_N' W_N x_N + w_N' x_N + \sum_{k=0}^{N-1}\left(\frac{1}{2}x_k' W_k x_k + w_k' x_k + x_k' F_k u_k + \frac{1}{2}u_k' \Lambda_k u_k + \lambda_k' u_k\right)$$

Equations (55) and (15) are similar since they are both quadratic functions. In fact Eq. (55) can be transformed to the form of Eq. (15) by expanding the quadratic terms. This results in the following relationship between the matrices of Eqs. (15) and (55).

(56) $W_k = \hat{W}_k$

(57) $w_k = -\hat{W}_k \tilde{x}_k$

(58) $F = 0$

(59) $\Lambda_k = \hat{\Lambda}_k$

(60) $\lambda_k = -\hat{\Lambda}_k \tilde{u}_k$

In the MATLAB statement we will input the data using the matrices and vectors in the tracking function (55) and then compute the matrices and vectors for the quadratic form (15) that was used to derive the algorithm which is implemented in the code.

The data for Eq. (55) which are taken from Kendrick (1982a) are as follows:

(61) $\tilde{x}_k = (1.0075)^k \begin{bmatrix} 387.9 \\ 85.3 \end{bmatrix}$

(62) $\tilde{u}_k = (1.0075)^k \begin{bmatrix} 110.4 \\ 147.17 \end{bmatrix}$

These two equations indicate that the desired paths for both the states and controls grow at approximately 3 percent per year or 0.75 percent per quarter over the time horizon covered by the model.

The priorities (penalty weights) in the objective function (55) are given below.

(63) $\hat{W}_N = \begin{bmatrix} 6.25 & 0 \\ 0 & 100 \end{bmatrix} \qquad \hat{W}_k = \begin{bmatrix} 0.0625 & 0 \\ 0 & 1 \end{bmatrix}$

(64) $\hat{\Lambda}_k = \begin{bmatrix} 1 & 0 \\ 0 & 0.444 \end{bmatrix}$

All of these priorities are the same (relative to the square of the size of the variables) except those for the terminal state variables in $\hat{W}_N$ where the priorities are 100 times as

great. This is done to represent the fact that politicians usually care more about the state of the economy in the period just before an election than they do at other times.

This completes the statement of the model. Now we are ready to incorporate both the mathematics and the model into the MATLAB representation.

5. The MATLAB Representation

In this section we discuss the MATLAB representation a few statements at a time. The complete listing is in Appendix 16A. The code statement is begun with the dimensions of the model. This version has seven time periods, two state variables and two control variables.

```
t = 7; n = 2; m = 2;
```

One of the nice features of the MATLAB language in comparison to older languages such as Fortran and C is that the dimensioning of matrices is done automatically by the code. Therefore it is not necessary to use something like

```
Dimension A(2,2), B(2,2), c(2,1)
```

Rather one can just input the matrices A and B and the vector c as shown below and the MATLAB system takes care of the memory management.

```
a = [0.914 -0.016;
     0.097  0.424];
b = [0.305 0.424;
    -0.101 1.459];
c = [-59.437;
    -184.766];
```

Notice that the semicolon is used to mark the end of a row in the matrix input.

Likewise we can input the initial conditions, x_0, for the state and the base values for the desired states and controls as vectors. The base values for the states are called `xtar` to indicate that they represent target values for the states, x. Likewise for the control targets which are called `utar`.

```
x0 = [387.9;
       85.3];
xtar = [387.9;
```

```
        85.3];
utar = [110.4;
        147.17];
```

Finally the criterion function matrices are input.

```
w = [0.0625 0;
       0    1];
wn = [6.25  0;
       0  100];
f = [0   0;
     0   0];
lambda = [1    0;
          0  0.444];
```

Now all the data have been input and we are ready to start the matrix Riccati loop. In preparation for doing so we need to initialize the Riccati matrix K_{k+1} and vector p_{k+1}. These are called `kold` and `pold` to distinguish them from K_k and p_k which will be called `knew` and `pnew` respectively. Since the Riccati loop proceeds from the last time period toward the first, period $k+1$ values are the old values and period k values are the new values.

```
%   The Riccati Loop

kold = wn;                      %   Boundary condition
pold = -wn*xtar*(1.0075)^t;     %   Boundary condition
```

Recall from Eqs (18) and (19), which were the terminal conditions for the Riccati equations, that

(30) $K_N = W_N$

(31) $p_N = w_N$

Also remember that we need to use input data for the tracking function Eq. (55) and transform it for use in quadratic form Eq. (15). Thus we need to use the transformations from Eqs. (56) and (57) which become for the terminal period

(56) $W_N = \hat{W}_N$

(57) $w_N = -\hat{W}_N \tilde{x}_N$

Substitution of Eq. (56) into Eq. (30) and Eq. (57) into Eq. (31) yields

(65) $K_N = \hat{W}_N$

(66) $p_N = -\hat{W}_N \tilde{x}_N$

In Eq. (66) the desired value for the state is based on the initial period target values grown over the time horizon covered by the model at a rate of 3 percent per year or 0.75 percent per quarter so that

(67) $\tilde{x}_N = (1.0075)^N \tilde{x}_0$

Substitution of Eq. (67) into Eq. (66) then provides the relationship which is used in the MATLAB representation namely

(68) $p_N = -\hat{W}_N \tilde{x}_0 (1.0075)^N$

This is written in MATLAB as

```
pold = -wn*xtar*(1.0075)^t;    %   Boundary condition
```

where `t` is the number of time periods in the MATLAB representation.

Next we need to compute and store the Riccati matrices K_k and for all time periods as we integrate backward from the terminal time period to the initial time period. So we need to store a series of (`n,n`) matrices. We do this by using a three dimensional matrix with dimensions (`n,n,t`) for the Riccati matrices K_k and (`n,t`) for the Riccati vectors p_k . This is specified in MATLAB with the statements.

```
kstore = zeros(n,n,t);       % storage for dynamic Riccati matrices
pstore = zeros(n,t);         % storage for dynamic Riccati vectors
```

So `kstore` will be our place to store all the Riccati matrices and `pstore` will be used to store the Riccati vectors. These matrices are filled with zeroes as they are created and later we will replace the zeros with the computed values.

Also, we need to create a place to store the optimal controls and states and that is accomplished with the MATLAB statements below. Here again the arrays are filled

with zeroes as they are created and then will be filled with the computed values later. This is done with the following statements.

```
u = zeros(m,t+1);
x = zeros(n,t+1);
```

The time dimension of these arrays is set to `t+1` rather than to `t` to accommodate the fact that the `x` array must holds values for period zero as well as for the last period.

As a final step before we begin the backward recursion we need to store the terminal values of `kold` in the matrix `t` of `kstore` and `pold in` column `t` of `pstore`.

```
kstore(:,:,t) = kold(:,:);
pstore(1:n,t) = pold;
```

This completes all the set up required before we began the backward recursion to compute the Riccati matrices and vectors. The loop itself is begun with the MATLAB statements

```
k = t-1;
while k >= 1;
```

Here the running index `k` is going to be used for time periods. It is initialized to `t-1` because we have already done the calculations for the terminal period `t` and are ready to do them for period `t-1`. Then the `while` command is used to indicate that the loop operation should continue so long as `k` is greater than or equal to `1`. Later on at the bottom of this loop we will find the statements

```
k = k-1;
end;                          %   End of the Riccati loop
```

that decrease `k` by one each time the calculation passes thorough the loop. Also the `end` statement indicates the point to which the calculation jumps once the condition in the `while` statement no longer holds true.

The next step is to compute the desired paths for the state and control vectors which are called `utark` and `xtark` for u target and x target respectively for all the time periods. Those variables are then used in turn in Eqs. (57) and (60), i.e.

(57) $w_k = -\hat{W}_k \tilde{x}_k$

(60) $\lambda_k = -\hat{\Lambda}_k \tilde{u}_k$

to compute w_k which is called `wsmall` and λ_k which is called `lambdas`.

```
utark = (1.0075^k).*utar;   %   Time dependent targets
xtark = (1.0075^k).*xtar;
wsmall = -w*xtark;
lambdas = -lambda*utark;
```

Now we are finally in position to compute the Riccati matrix using Eq. (32)

(32) $$K_k = A'K_{k+1}A + W_k - \left[A'K_{k+1}B + F\right]\left[B'K_{k+1}B + \Lambda_k'\right]^{-1}\left[F' + B'K_{k+1}A\right]$$

The representation in MATLAB of this equation is

```
knew = a'*kold*a+w-(a'*kold*b+f)*inv(b'*kold*b+lambda')*(f'+b'*kold*a);
```

This is a good demonstration of the power of MATLAB to represent a complex expression in a form that is very close to the mathematical representation. The differences between the mathematical and MATLAB representations are quickly apparent. For the inverse of a matrix enclosed in parentheses mathematics uses $(\ )^{-1}$ while MATLAB uses the function `inv( )`. MATLAB does not include the Greek alphabet so the mathematical symbol Λ is represented in MATLAB as `lambda`. Also, as was discussed earlier, the Riccati matrices K_k and K_{k+1} are represented as `knew` and `kold`, respectively.

Similarly the equation for the Riccati vectors in mathematics is

(33) $$p_k = -\left[A'K_{k+1}B + F\right]\left[B'K_{k+1}B + \Lambda_k'\right]^{-1}$$
$$\left[B'\left(K_{k+1}c\ + p_{k+1}\right) + \lambda_k\right] + A'\left(K_{k+1}c\ + p_{k+1}\right) + w_k$$

and its MATLAB representation is

```
pnew=
 -(a'*kold*b+f)*inv(b'*kold*b+lambda')*(b'*(kold*c+pold)+lambdas)+...
           a'*(kold*c+pold)+wsmall;
```

Note that the ... notation is use in MATLAB to signal to the complier that the rest of the equation continues on the following line.

Having now used `kold` and `pold` we need to transfer `knew` and `pnew` respectively to them for use in the next pass through the `while` loop. This is done with

```
kold = knew;                %   Setup next period
pold = pnew;
```

Then the Riccati matrix and vector for period `k` can be placed in the storage arrays that hold these matrices and vectors for all time periods, i.e.

```
kstore(:,:,k) = knew(:,:);
pstore(1:n,k) = pnew;
```

Now we are at the bottom of the backward loop and, as promised above, this loop is ended with a statement to decrease `k` by one and then to end the `while` loop.

```
    k = k-1;
    end;
```

No sooner do we finish to backward loop than it is time to start the forward loop with the statements.

```
    k = 0;
    xold = x0;
    sum = 0;

    while k <= t-1;
          utark = (1.0075^k).*utar;
          xtark = (1.0075^k).*xtar;

          wsmall = -w*xtark;
          lambdas = -lambda*utark;
```

After `k` is set to zero the state vector is initialized using `x0` and then the desired paths for the states and controls are computed and used to calculate w_k and λ_k which are represented in MATLAB as `wsmall` and `lambdas`. Notice that the `while` loop this time uses a less than or equal to. At the bottom of this forward `while` loop we find the statements

```
k = k+1;
end;
```

that increment the `k` and provide the close for the `while` loop.

Now we are to the stage where we want to make use of the Riccati matrices and vectors which we computed and stored away in the backward loop.

```
kold(:,:) = kstore(:,:,k+1);
pold = pstore(1:n,k+1);
```

The elements for the Riccati matrix are pulled from storage in `kstore` and the Riccati vector are pulled from `pstore`.

Once the K_k matrix is available in `kold` it can be used in the computation of the feedback gain matrix, G_k, as described in Eq. (34)

$$G_k = -\left[B'K_{k+1}B + \Lambda'_k\right]^{-1}\left[F_k + B'K_{k+1}A\right] \tag{34}$$

The MATLAB representation of this mathematical expression is

```
glarge = -inv(b'*kold*b+lambda')*(f'+b'*kold*a);
```

Here again we see how closely the mathematical and MATLAB representations parallel one another and how much this aids the user in being sure that the mathematics of the solution procedure are correctly mimicked in the computer code.

Similarly the mathematical expression for the feedback gain vector is

$$g_k = -\left[B'K_{k+1}B + \Lambda'_k\right]^{-1}\left[B'\left(K_{k+1}c + p_{k+1}\right) + \lambda_k\right] \tag{35}$$

and the MATLAB representation is

```
gsmall = -inv(b'*kold*b+lambda')*(b'*(kold*c+pold)+lambdas);
```

Once the feedback gain matrix and vector have been computed they can be used in the feedback rule (36) along with the state vector x_k to compute the control vector u_k.

(36) $$u_k = G_k x_k + g_k$$

```
uopt = glarge*xold + gsmall;
```

Finally, the system equation (37) is used along with x_k and u_k to compute x_{k+1}

(37) $$x_{k+1} = Ax_k + Bu_k + c$$

```
xnew = a*xold + b*uopt + c;
```

Next we need to compute the portion of the cost terms in the criterion function that are incurred during period k. This is done with the mathematical expression

(38) $$\frac{1}{2}\left(x_k - \tilde{x}_k\right)' W_k \left(x_k - \tilde{x}_k\right) + \frac{1}{2}\left(u_k - \tilde{u}_k\right)' \Lambda_k \left(u_k - \tilde{u}_k\right)$$

and the corresponding MATLAB expression

```
sum = sum + 0.5*(xold-xtark)'*w*(xold-xtark) + 0.5*(uopt-...
    utark)'*lambda*(uopt-utark);
```

The variable `sum` was set to zero at the top of the forward loop and is added to with each pass through the loop.

Then the values of the state and control vectors are stored

```
x(1:n,k+1) = xold;
u(1:m,k+1 )= uopt;
```

and `xold` is set to `xnew` for the next pass through the forward loop. Also the `k` index is incremented.

```
xold = xnew;
k = k+1;
```

Then the forward loop is closed with the statement

```
end;
```

One more small bit of clean up is required before we are through with the computations. We need to store the last state vector and to add on the portion of the cost function for the terminal period. This is done using a portion of the mathematics from Eq. (55), i.e.

$$\frac{1}{2}\left(x_N - \tilde{x}_N\right)' \hat{W}_N \left(x_N - \tilde{x}_N\right)$$

These two steps are represented in MATLAB as

```
x(1:n,t+1) = xold;
utark = (1.0075^k).*utar;
xtark = (1.0075^k).*xtar;
sum = sum + 0.5*(xold-xtark)'*wn*(xold-xtark);
```

The results for the optimal controls, states and criterion value are then printed with the statements

```
u = u';                 %   The optimal control vector
u
x = x';                 %   The optimal state vector
x
Criterion = sum         %   The value of the criterion function
```

In the preceding we have shown almost all the lines of the MATLAB code for this solution procedure and model. While it is nice to learn to code a few lines at a time, it is not nice to view it that way after you have gained some familiarity with it. Therefore Appendix 16A includes the complete listing of the MATLAB input file.

6. Experiments

The macroeconomic model used as an example here is delightfully small while you are learning how to represent and solve it in MATLAB, but after you have used it for

a bit you will discover that it has sharp limitations. Therefore, it is suggested that you modify it according to your taste while you are experimenting with it.

One limitation is that the coefficients on the control variable are so small that there is not much latitude to alter the state variable paths.

(52) $C_{k+1} = 0.914C_k - 0.016I_k + 0.305G_k + 0.424M_k - 59.437$

(53) $I_{k+1} = 0.097C_k + .424I_k - 0.101G_k + 1.459M_k - 184.766$

The coefficients of G_k in Eqs. (52) and (53) are 0.305 and -0.101 respectively. So you might want to increase the magnitude of these coefficients in order to make the model more responsive to fiscal policy. Alternatively, if you want more clout for monetary policy you might increase the size of the coefficients on M.

Also, some of you will not like the fact that government spending 'crowds out' investment, so you may want to change the coefficient on government expenditure in the investment equation from negative to positive.

Econometricians will be somewhat dismayed that we are suggesting that coefficients be altered since these are, after all, estimated from data and should not be changed willy-nilly. While we agree that one should be careful about empirical work, the spirit here is to learn about the dynamic response of the model. The estimation of this model on data from different time periods will indeed yield different parameter estimates and small changes in specification will also result in changes in the parameter values. So for purposes of this kind of experiment we would encourage the user to try some parameter modifications in order to see how that changes the optimal state and control variables.

Aside from these limitations, the model is reasonably good for becoming acquainted with the use of optimal control theory to determined macroeconomics policies. For this purpose the user is encouraged to alter either the desired paths of states and controls or to alter the priorities W_k and Λ_k to see how this affects the results. For example, some users will want to assign high priority to consumption and others will prefer to do so for investment. Some users will want to change the desired path for government expenditures so that it declines rather than rises and then provide a high priority weight in the (1,1) element in Λ_k to insure that this result is obtained. Some users will want to insure that the economy follows the desired paths in all periods and others will want instead to use high priorities only on the terminal period to be sure that the economy is in good shape just before the next election.

7. Further Readings

For a general treatment of dynamic programming methods and their applications to economics see Sargent (1987) and Adda and Cooper (2003). More advanced treatments can be found in Bertsekas (1995) and Stokey and Lucas (1989). For detailed derivations of the QLP, CE and OLF procedures and for projection-updating mechanisms, see Kendrick (1981) (2002). For a book on econometric and financial analysis with GAUSS, a language that is similar to MATLAB and which is widely used in econometrics, see Lin (2001).

Appendix 16A

MATLAB Representation of the Abel Model

```
% Title: Quadratic-Linear Tracking problem for Abel
% Program name: qlpabel.m

% Based on the Chapter 4 of Stochastic Control for Economic Models
% example by David Kendrick.
% GAUSS version by Hans Amman, modified to MATLAB by Huber Salas
% with subsequent changes by Miwa Hattori and David Kendrick
% to implement a deterministic,
% two-control version of the Abel (1975) model (Jan 2005)

% Computes the optimal cost-to-go, control and state vectors.

%
%   Preliminaries
%
t = 7; n = 2; m = 2;


a = [0.914 -0.016;
    0.097  0.424];
b = [0.305 0.424;
    -0.101 1.459];
c = [-59.437;
    -184.766];
x0 = [387.9;
       85.3];
xtar = [387.9;
       85.3];
utar = [110.4;
       147.17];
w = [0.0625 0;
         0  1];
wn = [6.25 0;
       0 100];
f = [0 0;
     0 0];
lambda = [1    0;
          0 0.444];
```

```
%
%   The Riccati Loop
%
kold = wn;                      %   Boundary condition
pold = -wn*xtar*(1.0075)^t;     %   Boundary condition

kstore = zeros(n,n,t);      % storage for dynamic Riccati matrices
pstore = zeros(n,t);        % storage for dynamic Riccati vectors

u = zeros(m,t+1);
x = zeros(n,t+1);

kstore(:,:,t) = kold(:,:);
pstore(1:n,t) = pold;


k = t-1;
while k >= 1;
  utark = (1.0075^k).*utar;   %   Time dependent targets
  xtark = (1.0075^k).*xtar;
  wsmall = -w*xtark;
  lambdas = -lambda*utark;

  knew = a'*kold*a+w-
      (a'*kold*b+f)*inv(b'*kold*b+lambda')*(f'+b'*kold*a);
                            %   Computing the Riccati matrices
  pnew =
   -(a'*kold*b+f)*inv(b'*kold*b+lambda')*(b'*(kold*c+pold)+lambdas)+...
             a'*(kold*c+pold)+wsmall;
                            %   Computing the tracking equation

  kold = knew;                %   Setup next period
  pold = pnew;
  kstore(:,:,k) = knew(:,:);
  pstore(1:n,k) = pnew;
  k = k-1;
end;                        %   End of the Riccati loop
```

```
%
%   The Forward loop
%
k = 0;
xold = x0;
sum = 0;

while k <= t-1;
  utark = (1.0075^k).*utar;
  xtark = (1.0075^k).*xtar;

  wsmall = -w*xtark;
  lambdas = -lambda*utark;

  kold(:,:) = kstore(:,:,k+1);
  pold = pstore(1:n,k+1);

  glarge = -inv(b'*kold*b+lambda')*(f'+b'*kold*a);
  gsmall = -inv(b'*kold*b+lambda')*(b'*(kold*c+pold)+lambdas);

  uopt = glarge*xold+gsmall;
  xnew = a*xold+b*uopt+c;
  sum = sum+0.5*(xold-xtark)'*w*(xold-xtark)
           +0.5*(uopt-utark)'*lambda*(uopt-utark);

  x(1:n,k+1) = xold;
  u(1:m,k+1) = uopt;
  xold = xnew;
  k = k+1;

end;                          %   End of the forward loop


%
%   The Last Period
%
x(1:n,t+1) = xold;
utark = (1.0075^k).*utar;
xtark = (1.0075^k).*xtar;
sum = sum+0.5*(xold-xtark)'*wn*(xold-xtark);
```

```
%
%   Print the solution
%
u = u';                         %   The optimal control vector
u

x = x';                         %   The optimal state vector
x

Criterion = sum                 %   The value of the criterion function
```

Part III

Special Topic: Stochastic Control

Special Topic: Stochastic Control

This third part of the book is different than the first two. The first two parts covered a wide variety of topics in a way that provides an introduction to the computational methods used in those fields. In contrast, this part zeroes in on a narrow area of computational economic research that is of particular interest to us. The area is the application of stochastic control theory methods to macroeconomic stabilization models. In the past we have done three kinds of work in this area: analytical (viz. Mercado (2004)), computational with the MATLAB software (viz. Amman and Kendrick (2003)) and computational with the Duali software (viz Amman and Kendrick (1999d)). Here we focus on the third of these three types of research since the Duali software provides a low-entry cost way to begin work in this field. However, the Duali software represents a sharp contrast to the software used in the first two parts of this book. The software systems used earlier in the book are all high quality commercial software. In contrast, the Duali software is experimental software that is under development by two of us (Amman and Kendrick).

The Duali software is intended to provide a point and click interface for a stochastic control program that can be used to solve models with a quadratic tracking criterion function, linear systems equations and stochastic specifications that may include additive and multiplicative noise terms, measurement errors and uncertainty about initial conditions. It is not a commercial product and has not had the extensive testing that is a part of such products. Rather it is an academic piece of software for which there is no support staff or help line. Also, the software has not yet even reached the "beta" stage and thus is prone to crashes. It therefore must be used with care since it can cause one to lose not only Duali input files but also input files for other applications that are running concurrently with Duali.

On the other hand, this software has been used successfully by many undergraduate and graduate students in classes at the University of Texas and has provided an easy "on ramp" for many students into the field of stochastic control. In addition, it has been used by a number of graduate students in developing some parts of their Ph.D. dissertation research. Therefore, we suggest that if you decide to move forward into this part of the book the gains may be substantial, but you should proceed with considerable caution.

If you choose to go forward, it is best to begin by making use of the User's Guide for the Duali software which can be found by going to the book web site at

http://www.eco.utexas.edu/compeco

and proceeding to the Software section of the web site. The User's Guide will introduce you to the capabilities of Duali and take you through the steps to setup and solve a simple macroeconomic model. Once you have done that, the material in this third part of the book will follow logically.

Chapter 17

Stochastic Control in Duali[39]

In an earlier chapter we presented the Hall and Taylor macroeconomic model, a standard nonlinear dynamic model for an open economy with flexible exchange rates. There we represented and simulated the model in GAMS, and we also introduced a basic form of optimal policy analysis. Working with the same model, in this chapter we will take some steps forward in the realm of policy analysis providing an introduction to the field of stochastic control.

A stochastic control problem can be posed as one in which a policymaker, manipulating a set of control variables, tries to influence the dynamics of an economic system in order to achieve some targets. For example, in a macroeconomic setting, the policymaker may use some controls - policy variables such as the money stock or government expenditure - to influence the behavior of the economy in order to maintain some target variables such as unemployment and inflation as close as possible to their desired paths. The economic model is usually represented in state-space form, that is, as a first-order system of dynamic equations. The policymaker has an objective function - usually a quadratic one - which specifies the target variables, the desired paths and the relative weights put on the achievement of each target.

The solution of deterministic and stochastic control problems quickly becomes very involved. Thus, to make our task feasible, we have to rely on computational methods and specialized software. Duali[40] is software that can receive as inputs the desired paths and corresponding weights for target and policy variables and the state-space representation of the economic model. It can then be used to generate simulation results and to compute the optimal policy rule and the implied solution paths for policy and target variables using the methods described earlier in the chapter on Dynamic

---

[39] This Chapter draws extensively on both the verbal and mathematical development in Mercado and Kendrick (1999). Kluwer Academic Publishers have kindly granted us permission to reuse here substantial materials from our previously published paper.

[40] See Amman and Kendrick (1999a). Special care should be taken when doing the experiments in this book which use Duali. If you have not already read about the Duali software in the introduction to this part of the book, please go back and do so.

Optimization. In what follows, we will use Duali to perform first deterministic and then stochastic control experiments with the state-space representation of Hall and Taylor's model.

1. The Hall and Taylor Model in State-Space Form

Much undergraduate study of macroeconomics makes use of dynamic *nonlinear* models in levels; for example the levels of government expenditure and the money supply are used to determine the levels of consumption, investment, output, interest rates and net exports. In contrast, much empirical macroeconomic research centers on dynamic *linear* models in percentage deviations of variables from their steady state values. In these empirical models one alters the percent deviation of government expenditures and the money supply from their steady state levels and analyzes the resulting deviations of consumption, investment, output, interest rates and net exports from their steady state levels. However, the bridge between these two types of models is frequently not clear.

Therefore, for this chapter we have begun with the dynamic nonlinear Hall and Taylor model in levels that we used with GAMS earlier in the book and have transformed it into a linearized model in percentage deviations of variables from their steady state values in a similar fashion to the approach we used in the chapter that includes the Johansen type CGE model. Also, we will use here a four-equation linear version of the original Hall and Taylor twelve-equation nonlinear model which will capture the essential behavior of the original model. Thus, our four-equation model's variables will be

Endogenous Variables

Y^* = GDP
R^* = Real Interest Rate
$plev^*$ = Domestic Price Level
E^* = Nominal Exchange Rate

Policy Variables

M^* = Money Stock
G^* = Government Expenditure

Exogenous Variables

$plevw^*$ = Foreign Price Level

YN^* = Potential GDP

The asterisks indicate "percent deviations", for example, Y^* is the percent deviation of output from its steady state value. This variable structure is one of the most common ways in which textbook macroeconomic models are presented. To transform the original twelve-equation nonlinear model we first collapsed it, by equation substitution, into a four-equation version. We then linearized these equations and represented the resulting model in matrix form. Next we solved the model for its reduced form representation, obtaining a third-order system of difference equations. Finally, we reduced that system to a first-order system, that is, to its state-space form. Details on these transformations are provided in Appendix E.

The model's state-space representation is

$$x_{k+1} = Ax_k + Bu_k + Cz_k \tag{1}$$

where x is an augmented state vector defined as

$$x = \begin{bmatrix} X \\ XL \\ XLL \end{bmatrix}$$

where

$$X = \begin{bmatrix} Y^* \\ R^* \\ plev^* \\ E^* \end{bmatrix}$$

and where XL and XLL are equal to the vector X lagged once and twice respectively. We define the x vector in this way by augmenting the original state vector with lagged values in order to reduce the linearized model from a third-order representation to a first-order representation (see Kendrick (2002), Ch. 2). The control vector and the exogenous variables vector are defined as

$$u = \begin{bmatrix} M^* \\ G^* \end{bmatrix} \qquad z = \begin{bmatrix} YN^* \\ plevw^* \end{bmatrix}$$

.

Also, the parameter matrices in the system equations are

$$A = \begin{bmatrix}
-0.346 & 0 & -0.606 & 0 & 0 & 0 & 0.087 & 0 & 0 & 0 & 0.087 & 0 \\
7.811 & 0 & 13.669 & 0 & 0 & 0 & -1.953 & 0 & 0 & 0 & -1.953 & 0 \\
0.8 & 0 & 1.4 & 0 & 0 & 0 & -0.2 & 0 & 0 & 0 & -0.2 & 0 \\
1.154 & 0 & 2.019 & 0 & 0 & 0 & -0.288 & 0 & 0 & 0 & -0.288 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0
\end{bmatrix}$$

$$B = \begin{bmatrix}
0.433 & 0.231 \\
-9.763 & 4.386 \\
0 & 0 \\
-2.442 & 1.097 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0
\end{bmatrix} \qquad C = \begin{bmatrix}
0.346 & 0 \\
-7.811 & 0 \\
-0.800 & 0 \\
-1.154 & 1 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0
\end{bmatrix}$$

where each of these matrices values is derived from the corresponding combination of parameter values in the original twelve-equation nonlinear Hall and Taylor model. See Appendix E for these derivations.

2. Introduction to Optimal Policy Analysis Methods with Duali

In an earlier chapter we used GAMS to study the responses of Hall and Taylor's model to changes in the policy variables. Optimal policy analysis employs a sort of "reverse" analysis. It begins by posing this question: how should policy variables be set in order for the target variables to follow pre-specified paths?

The most popular way of stating this problem is as a Quadratic Linear Problem (QLP). We have already introduced this type of problem in the Thrift Model chapter and the Dynamic Optimization chapter. In formal terms, we express our problem here as one of finding the controls $(u)_{t=0}^{N}$ to minimize a quadratic "tracking" criterion function J of the form:

$$(2)\ J=\frac{1}{2}\left[x_N-\tilde{x}_N\right]' W_N\left[x_N-\tilde{x}_N\right]+\frac{1}{2}\sum_{k=0}^{N-1}\left(\left[x_k-\tilde{x}_k\right]' W_k\left[x_k-\tilde{x}_k\right]+\left[u_k-\tilde{u}_k\right]'\Lambda_k\left[u_k-\tilde{u}_k\right]\right)$$

subject to the state-state representation of the economic model given by Eq. (1), where $\tilde{x}$ and $\tilde{u}$ are desired paths for the state and controls variables respectively and W and Λ are weighting matrices for states and controls respectively.

The quadratic nature of the criterion function implies that deviations above and below target are penalized equally, and that large deviations are more than proportionally penalized relative to small deviations. This particular form of the criterion function is not the only possible one, but is the most popular.[41]

For simplicity in the following we will drop the asterisk from the variables. Thus we will use Y, R, $plev$ and E instead of Y^*, R^*, $plev^*$ and E^* to indicate the state variables. However, we are referring to the variables as percent deviations rather than as levels.

[41] For a discussion of the properties of different criterion functions, see Blanchard and Fischer (1989), Chapter 11.

We will assume that the policy goal is to stabilize *Y*, *R*, *plev* and *E* around steady-state values (that is, around zero). High and equal weights[42] will be put on stabilizing *Y* and *plev*, lower and equal weights on *R* and *E*, and even lower and equal weights on the policy variables *M* and *G*. Neither the desired paths nor the weighting matrices (shown below) will vary with time.

$$W = \begin{bmatrix} 100 & & & & & & & & & \\ & 50 & & & & & & & & \\ & & 100 & & & & & & & \\ & & & 50 & & & & & & \\ & & & & 0 & & & & & \\ & & & & & 0 & & & & \\ & & & & & & 0 & & & \\ & & & & & & & 0 & & \\ & & & & & & & & 0 & \\ & & & & & & & & & 0 \end{bmatrix} \qquad \Lambda = \begin{bmatrix} 25 & \\ & 25 \end{bmatrix}$$

Let's assume, for example, that the economy is going through a recession provoked by a temporary adverse shock to net exports that causes *Y* to be 4% below its steady-state value. Given the weight structure adopted above, what would be the optimal paths for government expenditure (*G*) and the money supply (*M*) in order to bring the economy back to its steady-state? How do the optimal paths for the state variables compare against what would be the autonomous response of the system to that kind of shock? To answer these questions, we perform two experiments: (1) an experiment to obtain the optimal paths and (2) an experiment to get the autonomous response of the

[42] There is a conceptual difference between the weights used here and those that arises when the variables of interest are in levels rather than in percent deviations and also where the variables are expressed in different units of measurement. For instance, if GDP is measured in dollars and prices are measured by an arbitrary price index, equal weights on these two variables will probably imply different policy priorities and vice versa. Since all variables in the state-space representation of Hall and Taylor's model are in percent deviations from steady-state, weighs and priorities can be considered as equivalent within certain limits. However, it should be clear that, for example, an interest rate 50% below steady-state values is something feasible, while a level of GDP 50% below steady-state is not. In such a case, there is not an analogy between weights and priorities. See Park (1997).

economy. To run this simulation, use program `ht-01.dui` making the appropriate changes. (See the "Model Description" item in the "Specification" menu in Duali once the `ht-01.dui` file is opened in the application.)

To perform the first experiment in Duali, we have to set the problem as a deterministic one, set all the desired paths for states and controls equal to zero, impose the corresponding weights on states and controls, set an initial value for Y equal to -0.04, and solve the problem. Let's see in more detail how to do this.

Below is the initial screen of the Duali software. The File and Edit menus are standard. The Specification and Data menus contain sub-menus related to the structure of the problem to be solved. The Solve menu presents options for different solution methods and the Results menu enables one to display the tables and graphs of the results. The Transformations menu contains several options to change the original structure of the problem, and the Preferences menu contains options related to the format of the display of results and to some specific types of experiments.

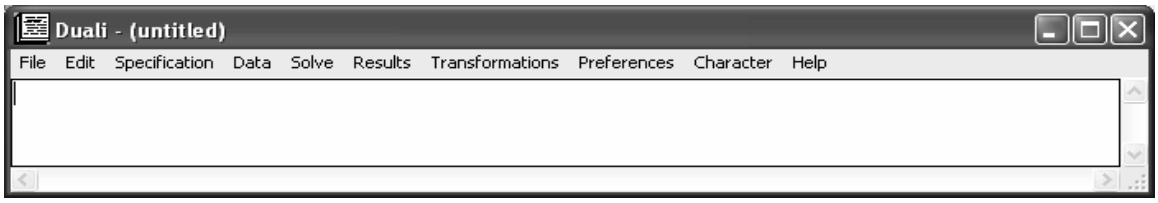


Figure 17.1 Duali Main Window

We begin by opening the `ht-01.dui` file using the File menu. Then select Specification:Stochastic Terms and notice that the problem is set as deterministic, as shown in Fig. 17. 2.

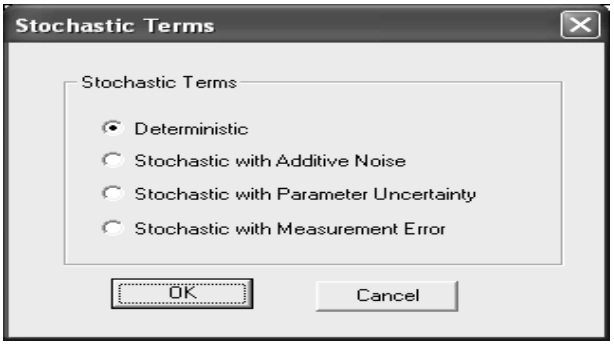


Figure 17.2 Stochastic Terms Dialog Box

Then, from the Specification:Functional Form option we obtain the dialog box shown below.

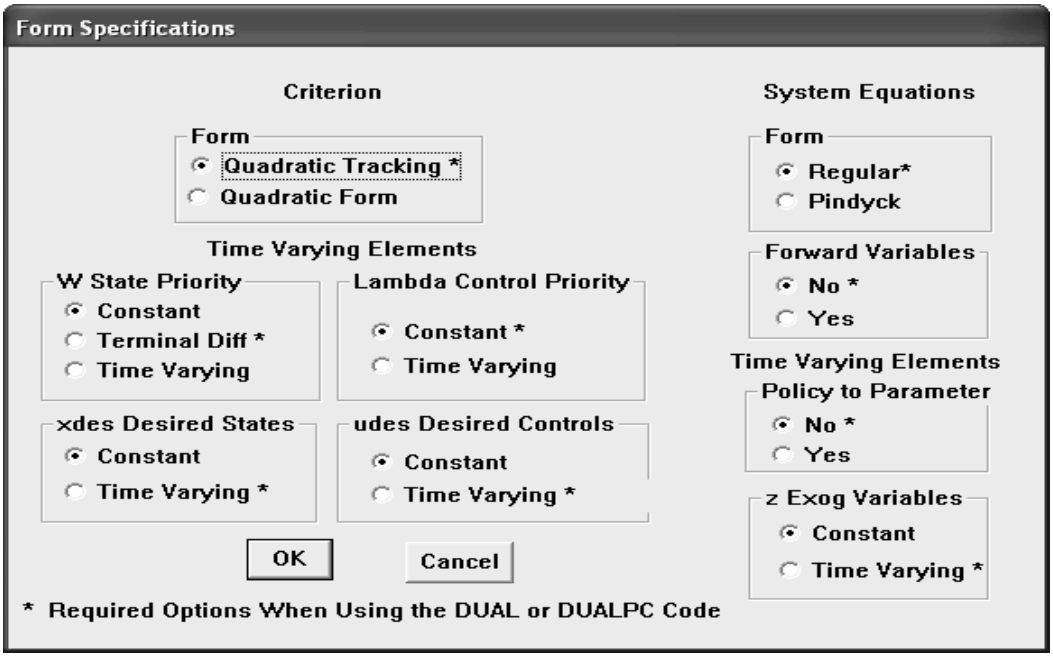


Figure 17.3 Form Specifications Dialog Box

In Fig. 17. 3 look at the Criterion side of the dialog box and at the Form section in that side. There one can see that the problem is a Quadratic-Tracking problem. In fact, we will try to minimize deviations of target variables from zero, since the model variables are already expressed in percent deviations from steady-state values. The W State Priority and Lambda Control Priority sections show that the weights on state and control variables will be constant, that is, the same value for all periods. Desired state and control variables will also be constant (all zeroes). The right hand side of the dialog box shows the specification for the System Equations. In particular, the Form section shows that the problem is written in (1) regular form, that is the standard state-space representation, (2) it does not contain forward variables (as will be the case of models with rational expectations discussed later) and (3) the policy variables do not affect the model parameters in this particular model. Finally, the exogenous variables remain constant over the time periods.

From the Data:Size menu we obtain the dialog box below.

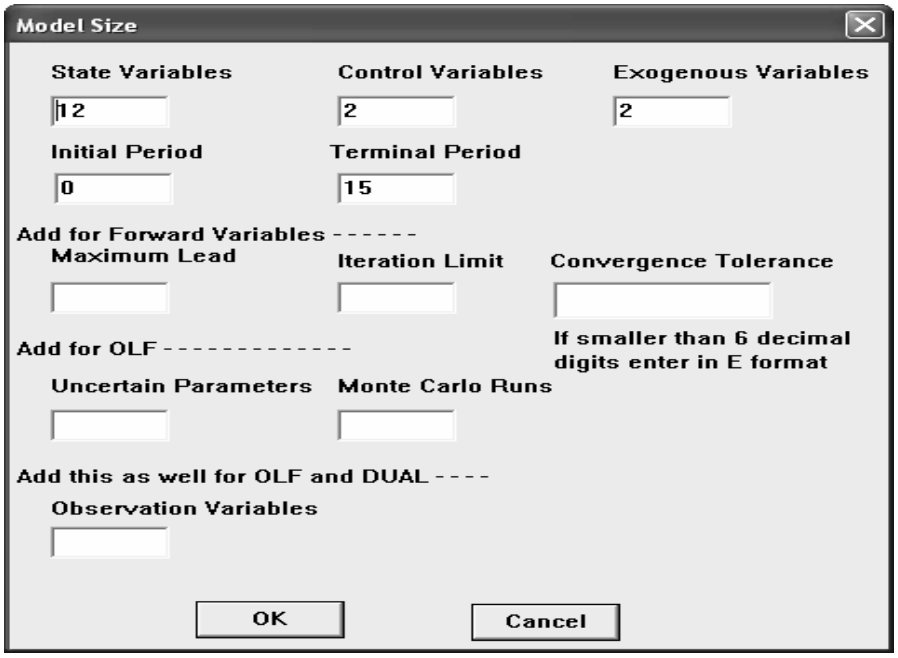


Figure 17.4 Model Size Dialog Box

The model is specified as containing twelve state variables (actually, four contemporaneous and eight lagged), two control variables and two exogenous variables, and the simulation covers sixteen periods.

The Data:Acronyms menu option contains the assignment of labels to the model variables and to the time periods. The Data:Equations section contains the numerical values for matrices A, B, C and for the initial state variable values while the Data:Criterion section contains the values for the W and Λ weighting matrices and the desired paths values for state and control variables.

Choosing the menu option Solve:QLP the problem is solved as a Quadratic Linear Problem using the solution procedure described in the chapter on Dynamic Optimization. The numerical results are then displayed automatically. The Results menu options allow us to define different display, plotting and printing options.

The results of this experiment to obtain the optimal states are shown in Figure 17.5. Also, the graphs in that figure show the autonomous state and control paths. In order to obtain the autonomous path of the system we impose zero weights on the state variables, very high and equal weights on the controls and, as in the first experiment above, set an initial value for Y equal to -0.04. This has the effect of leaving the state variables free to take on any values while restricting the policy variables not to deviate from their steady state values.

The vertical axes in Fig. 17. 5 show the percent deviations from steady-state values while the horizontal axes show the time periods. In these plots a value of 0.02 means "2% above steady-state". It does not mean "2% increase with respect to the previous period". Thus, a 10% permanent increase in *M* means that the money stock is increased by 0.1 at the initial period and kept constant at the new level from then on. Since all variables (endogenous, policy and exogenous) are in percent deviations, their steady-state values are all zeroes.

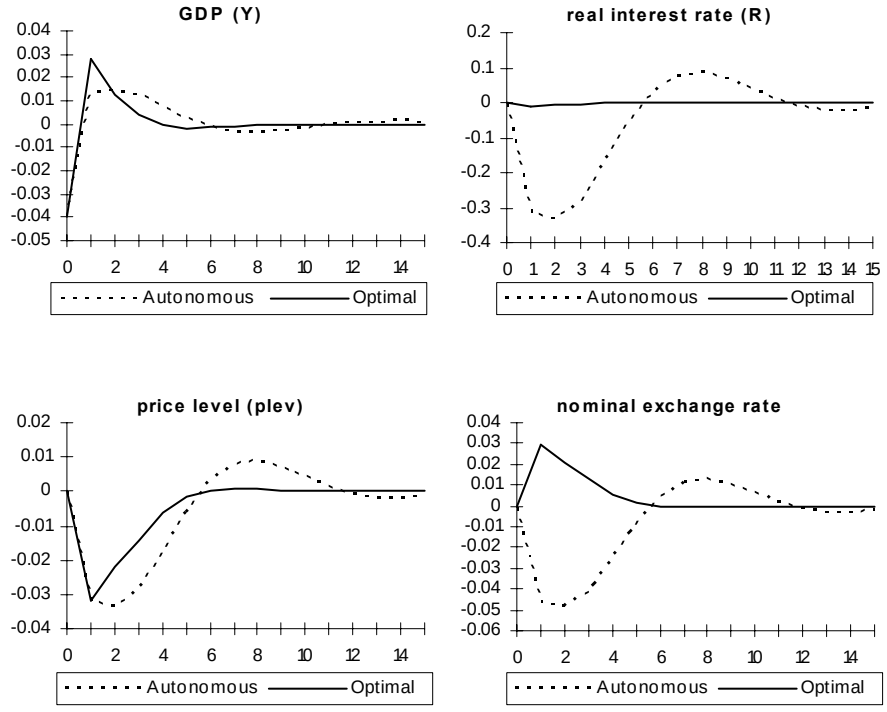


Figure 17.5 Autonomous Response vs. Optimal Control Experiments

The optimal solution paths for the states outperform the autonomous responses of the system for all four target variables. This comes as no surprise, though it may not always be the case. Indeed, remember that the optimal solutions are obtained from the minimization of an overall loss function. On some occasions, depending on the weight structure, it may be better to not do as well as the autonomous response for some targets in order to obtain more valuable gains from others.

Why does the autonomous path of the economy display the observed behavior? Here is how Hall and Taylor explain it: [43]

"With real GDP below potential GDP after the drop in net exports, the price level will begin to fall. Firms will have found that the demand for their products has fallen off and they will start to cut their prices (...). The lower price level causes the interest rate to fall.[44] With a lower interest rate, investment spending and net exports will increase.[45] The increase in investment and net exports will tend to offset the original decline in net exports. This process of gradual price adjustment will continue as long as real GDP is below potential GDP."

What explains the observed optimal path of the four variables of interest? We can see in Fig. 17. 5 that *Y* is brought up very quickly, going from 4% below steady-state to 3% above steady-state and then decays slowly to its steady-state value. This performance could be attributed to the more than 6% increase in *G* that can be observed in the optimal policy variables' paths (Figure 17.6).

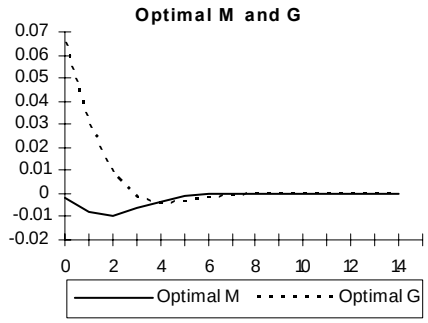


Figure 17.6 Optimal Policy Variables Paths

Meanwhile in Fig. 17. 5, *R* experiences almost no variation when compared to the big drop of almost 35% implied by the autonomous behavior of the system. Once again, the increase in *G* puts an upward pressure on the interest rate, thus keeping it from

[43] Hall and Taylor (1997), page 232.

[44] Since less money is demanded by people for transactions purposes.

[45] Since the price level falls much less than the real interest rate during the first periods of the adjustment, the nominal exchange rate has to fall too, as can be derived from equation 9 in the original Hall and Taylor's model. This implies that the real exchange rate will fall, then causing net exports (see equation 10) to rise.

falling. Finally, the nominal exchange rate has to go up to compensate for the fall in prices, given that the real interest rate does not change much.

We can also see in Fig. 17. 6 that monetary policy plays a minor role when compared to fiscal policy.[46] Even though we put the same weights on both variables, government expenditure appears to be more effective in bringing the economy out of its recession given the weight structure we put on the target variables.

It is interesting to analyze the different combinations of behavior of variables that the policy maker can achieve given a model and a criterion function. The curve showing those combinations is known as the policy frontier.[47] For instance, we may want to depict the trade-off between the standard deviations of *Y* and *plev* in Hall and Taylor's model when, as above, *Y* is shocked by a negative 4% in period zero. To obtain the corresponding policy frontier, we have to vary the relative weights on *Y* and *plev*, perform one simulation for each weight combination and compute the corresponding standard deviations. The results of six such experiments, keeping the same weights on the remaining states and controls as in the above simulation, are shown in Table 17.1 and Figure 17.7.

Experiment	**Weight on *Y***	**Weigh on *plev***	**STD *Y***	**STD *plev***
1	100	0	0.0479	0.0500
2	80	20	0.0489	0.0466
3	60	40	0.0499	0.0440
4	40	60	0.0509	0.0419
5	20	80	0.0520	0.0401
6	0	100	0.0531	0.0386

Table 17.1 Optimal Policy Frontier

[46] Notice that the optimal values for the policy variables are computed for periods 0 to 14 only. Given that we are working with a state-space representation of the model, policy variables can only influence the next period state variables. That is, the controls at period 0 are chosen, with a feedback-rule, as a function of period 0 states, but they determine period 1 states, and so on. See Kendrick (1981).

[47] See Hall and Taylor (1997), Chapter 17.

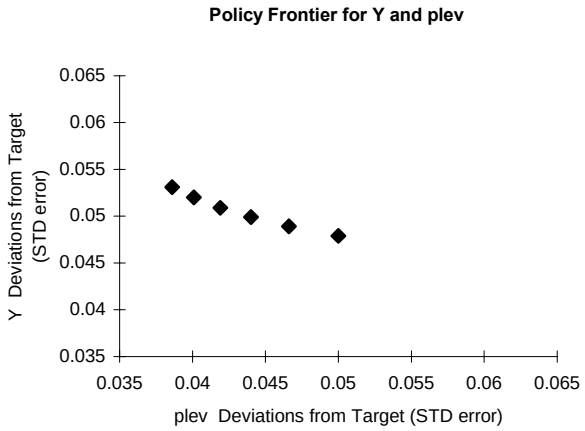


Figure 17.7 Optimal Policy Frontier Graph

The policy frontier for *Y* and *plev* is clearly shown in the graph above, where each diamond represents the result of an experiment. The higher the weight on *Y* relative to that of *plev*, the lower its standard deviation, and vice versa. The flatness of the curve indicates that it is easier to achieve a reduction in the percent deviation from target for plev than for *Y*. Of course, shape and location of this particular policy frontier are conditional on the weight structure imposed on the model's other variables. For example, if we increase the weigh on the policy variables, the policy frontier will shift up and to the right, farther away from the origin (the (0,0) point of zero deviations for *Y* and *plev*). This will be due to the more restricted possibilities for actively using the policy variables to reach the targets for *Y* and *plev*.

3. Stochastic Control

We will now begin to take uncertainty into account. Indeed, macroeconomic models are only empirical approximations to reality. Thus, we want to consider that there are random shocks hitting the economy every time period (additive uncertainty), that the model parameters are just estimated values with associates variances and covariances (multiplicative uncertainty), and that the actual values of the model's variables and initial conditions are never known with certainty (measurement error). [48]

[48] See Kendrick (1981).

Stochastic control methods artificially generate a dynamic stochastic environment through random shocks generation. They use specific procedures for choosing the optimal values for each period policy variables: Certainty Equivalence (CE) when there is additive uncertainty only, Open Loop Feedback (OLF) when there is parameter uncertainty, and DUAL (adaptive control) when there is active learning. Also there are specific mechanisms of updating of parameter estimates. In that way, these methods allow us to perform sophisticated simulations.

In this section, we will perform experiments incorporating some forms of additive and multiplicative uncertainty into Hall and Taylor's model. We will proceed in three steps. First, we will analyze the differences in qualitative behavior of the policy variables when some different procedures for choosing their optimal values are used (specifically, CE versus OLF w/o update). Second, we will compare the quantitative performances of the CE and OLF procedures within artificially generated stochastic environments including passive learning mechanisms. Finally, we will compute an optimal policy frontier.

Years ago William Brainard (1969) showed that, for a static model, the existence of parameter uncertainty causes the optimal policy variable to be used in a more conservative way as compared to the case of no parameter uncertainty. However, this finding cannot be translated to the case of dynamic models. The existence of dynamics makes the situation much more complex and opens new possibilities for policy management. One of the earliest applications of an OLF procedure in a dynamic setting was by Tinsley, Craine and Havenner (1974). Some analytical results have been provided by Chow (1973), Turnovsky (1975), Shupp (1976), Craine (1979) and more recently by Mercado and Kendrick (2000) and Mercado (2004) in connection with the qualitative behavior of the policy variables when the OLF procedure is used in a model with one state and one or two controls. There are no straightforward theoretical results for the case of models with several states and controls.

As shown earlier in the chapter on Dynamic Optimization, the procedure for choosing the controls in the presence of parameter uncertainty (OLF) differs from the standard deterministic QLP procedure or its "certainty equivalent" (CE) in that in the first case the variances and covariances associated with the model parameters have to be taken into account.

To illustrate some possible outcomes, and to show a first contrast between patterns of behavior generated by QLP and OLF w/o update procedures[49], we will perform an experiment with Hall and Taylor's model. As in the previous section, we will assume that Y is 4% below its steady-state value at time zero and we will keep the same weight structure and desired paths. We will also assume that there is uncertainty in connection with six out of the eight control parameters in the B matrix, and that the standard deviation of each of these parameters is equal to 20%.

To carry out the experiment, we use the program `ht-02.dui`. This program is basically the same as `ht-01.dui`, with some changes that we discuss below. From the Specification:Stochastic Terms option we see that the problem is set as stochastic with parameter uncertainty.

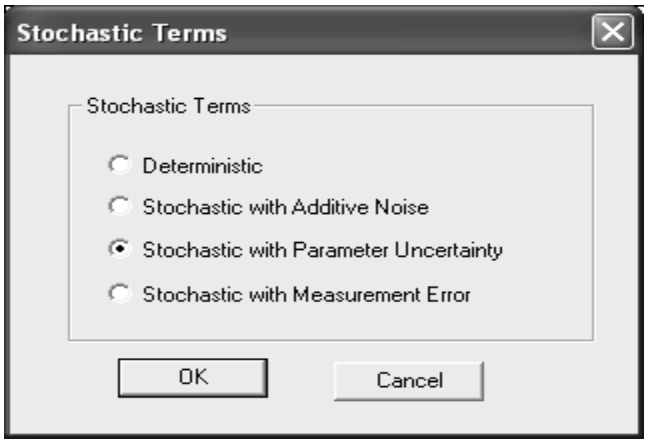


Figure 17.8 Stochastic Terms Dialog Box

Then, from the Data:Size option, we see that we defined 6 uncertain parameters and use 1 Monte Carlo run, as shown in the dialog box below.

[49] For a detailed discussion of the OLF without update procedure see Ch. 5 "Open Loop Feedback without Update" in Amman and Kendrick (1999a).

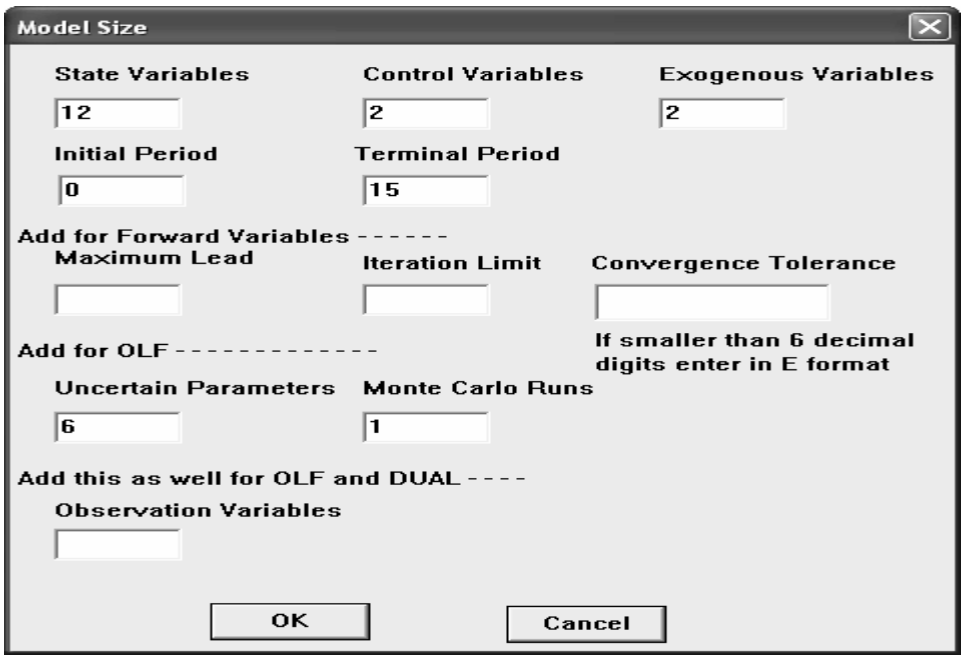


Figure 17.9 Model Size Dialog Box

From the Specification:Source of Random Terms main menu option, we select the Read In option, as shown below.

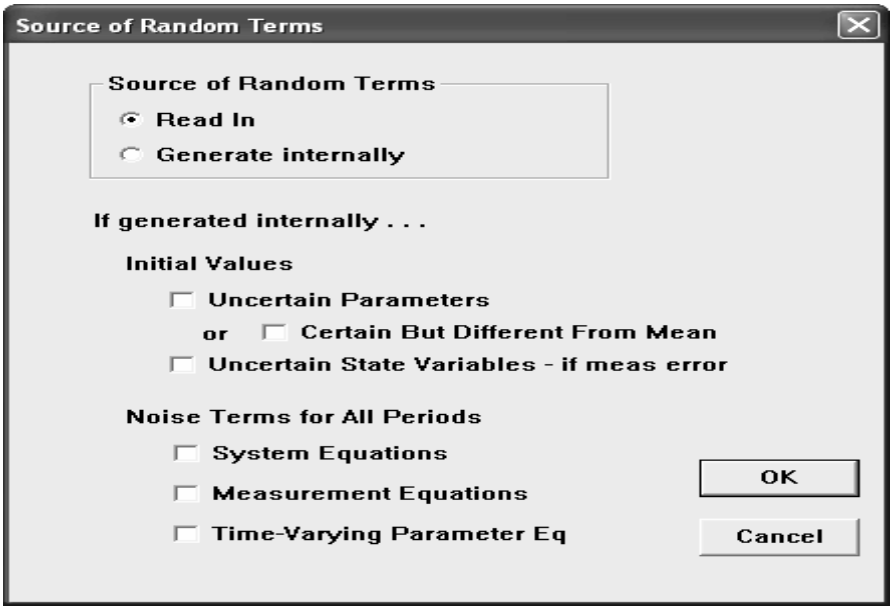


Figure 17.10 Source of Random Terms Dialog Box

However, we set those random terms all equal to zero. To do so, we go to the Data:Additive Noise Terms main menu option and, as shown below, select the XSIS option.

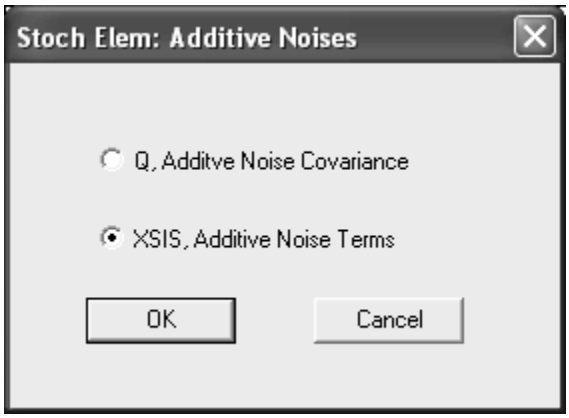


Figure 17.11 Stochastic Elements Additive Noises Dialog Box

When doing so, a dialog box containing the matrix of additive noise terms will display, and we will see that all its element are set to zero.

The information related to the uncertain parameter is provided in Duali by means of one vector and two matrices. The theta vector of the initial values of uncertain parameters (TH0) contains the uncertain parameters values. The matrix that indicates which parameters in the model are treated as uncertain (ITHN) provides a mapping from the position in the TH0 vector to position in the system equations matrices. The first column indicates the matrix (0 for the *A* matrix, 1 for the *B* matrix and 2 for the *c* vector) and the second and third columns indicate the row and column number of the parameter in the matrix. Finally, SITT0 is the variance-covariance matrix corresponding to the uncertain parameters.

(3)
$$TH0 = \begin{bmatrix} b_{11} = 0.433 \\ b_{12} = 0.231 \\ b_{21} = -9.763 \\ b_{22} = 4.386 \\ b_{41} = -2.442 \\ b_{42} = 1.097 \end{bmatrix}, ITHN = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 1 & 2 & 1 \\ 1 & 2 & 2 \\ 1 & 4 & 1 \\ 1 & 4 & 2 \end{bmatrix},$$

$$SITT0 = \begin{bmatrix} 0.00749 & & & & & \\ & 0.00213 & & & & \\ & & 3.81264 & & & \\ & & & 0.76947 & & \\ & & & & 0.23853 & \\ & & & & & 0.04813 \end{bmatrix},$$

All three matrices will remain constant during the simulation. The elements in SITT0 are computed by taking 20% of the corresponding element in TH0 and then squaring the result. Thus, for the b_{11} coefficient this is

$$[(0.2)\ (0.433)]^2 = 0.00749.$$

From the Data:Parameter Uncertainty menu option we obtain the dialog box below. When selecting each of the first three options, the corresponding vector or matrix will be displayed.

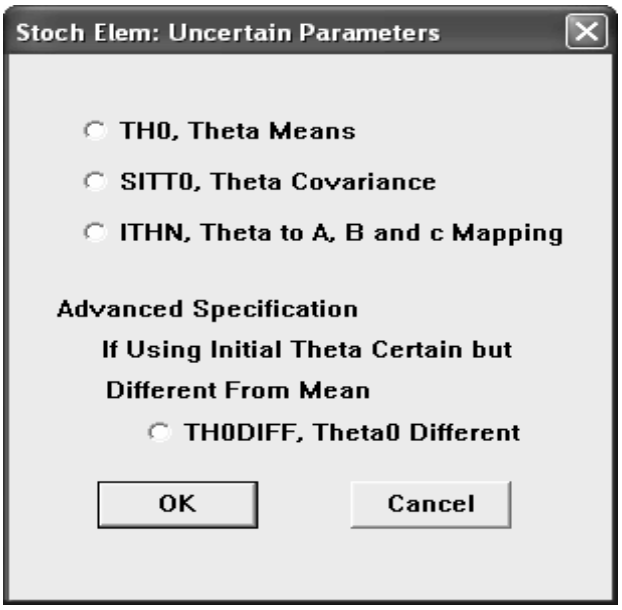


Figure 17.12 Stochastic Elements Uncertain Parameters Dialog Box

The graphs below in Figure 17.13 show the results obtained for government expenditure and for the money supply when selecting the main menu option Solve: OLF

(w/o update). They also contrast these results with those corresponding to the deterministic (QLP) solution as obtained in section 2 of this chapter using the program `ht-01.dui`.

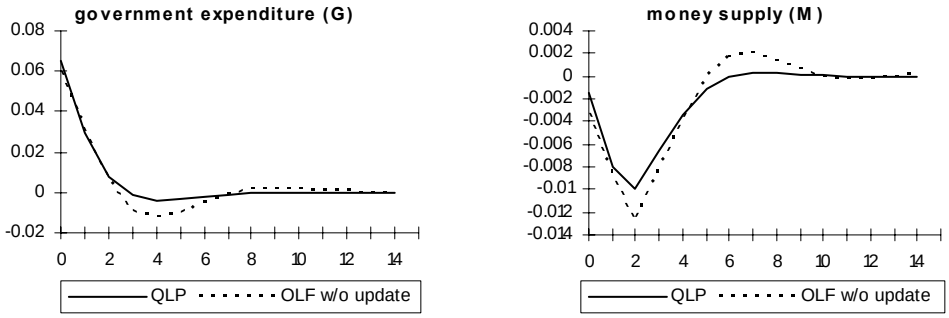


Figure 17.13 Optimal Policy Variables Paths (QLP vs. OLF w/o update)

As can be seen in the graphs above, the use of government expenditure is slightly more "cautious" with the OLF w/o update procedure in the first few periods. This is in line with the Brainard result mentioned before. However, the reverse is true for the case of the money supply, which is used "more aggressively" with OLF w/o update. Thus, we can see how going from a univariate to a multivariate setting may have important consequences, as is also the case of a change from static to dynamic models.

It is interesting to explore the consequences of increasing the level of uncertainty of the model parameter's corresponding to one of the policy variables. For example, let's assume that we now double the standard deviation of the parameters corresponding to government expenditure (parameters b_{12} from 0.00213 to 0.00853, b_{22} from 0.76947 to 3.07791 and b_{42} from 0.04813 to 0.19254 while leaving the other elements of SITT0 unchanged) i.e. increasing the variance of these three parameters that are associated with government expenditures from 20% to 40%. Then, the SITT0 matrix becomes:

(4)
$$SITT0 = \begin{bmatrix} 0.00749 & & & & & \\ & 0.00853 & & & & \\ & & 3.81264 & & & \\ & & & 3.07791 & & \\ & & & & 0.23853 & \\ & & & & & 0.19254 \end{bmatrix}$$

The graphs below contrast the behavior of the policy variables for this experiment (named OLF w/o update-B) against their behavior shown by the same variables in the experiment analyzed above (named, as above, OLF w/o update). To run this experiment, use file `ht-02.dui`, introducing the corresponding changes in the SITT0 matrix.

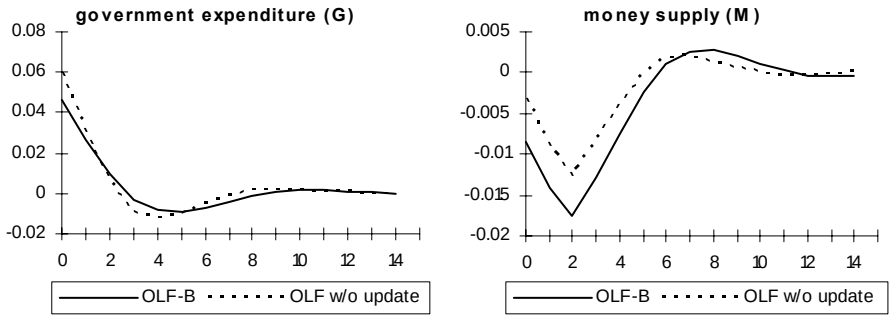


Figure 17.14 Optimal Policy Variables Paths (Increased Uncertainty)

As one could expect, the increase in the relative uncertainty of government expenditure parameters induces a more cautious use of that policy variable, at least during the first periods. At the same time the money supply, now with a relatively lower associated uncertainty, is used more actively, also during the first periods. Though these findings seem plausible, they do not reflect any theoretical result, since such results are not yet available. As with the previous experiment, we could perhaps find different results for a different model.

4. Stochastic Control with Parameter Updating

We will now move towards a more complex stochastic environment. As in the previous section, we will assume that that some of the model parameters are uncertain, but now we will also assume that the model is constantly shocked by additive noise, that the true model is not known to the policy maker, and also that a passive-learning process takes place. We will perform several Monte Carlo runs to contrast the performance of two procedures: CE and OLF.

The general structure of each Monte Carlo run will be as follows. At time zero, a vector of model parameters will be drawn from a normal distribution whose mean and variances are those of matrices TH0 and SITT0. Then, at each time "t", we will have:

1) random generation of a vector of an additive shocks
2) computation of the optimal controls for periods k to N (terminal period)
3) propagation of the system one period forward (from period k to period k+1) applying the vector of controls (for period k only) computed in step 2.
4) updating of the next period parameter estimates (both means and variance-covariance elements)

For choosing the optimal control at each period (step 2) we will use either a Certainty Equivalence (CE) procedure or, alternatively, an Open Loop Feedback procedure (OLF). For the projection-updating mechanism (step 4) we will use a Kalman filter.

Thus, each Monte Carlo run begins with a vector of parameter estimates that is different from their "true" value. Using this parameter vector, the policy maker computes (with a CE or an OLF procedure) the optimal values of the controls, and then she applies those values corresponding to time k only. However, the response of the economic system (its forward movement from time k to time k+1) will be generated by the computer using the "true" parameter values which are unknown to the policy maker. Then, at period k+1 a new observation is made of the state vector, which is used to compute updated parameter estimates with a Kalman filter. After a number of time periods, the sequence of updated estimates should begin to converge to their "true" value.

As in the previous section, we will assume that there is uncertainty in connection with six of the control parameters in the B matrix, and that the standard deviation of each of these parameters is equal to 20%. Then, matrices TH0, SITT0 and ITHN will be the same as in Eq (3). We will also assume that GDP (Y) and the price level (plev) are hit by additive shocks with 2% standard deviation, while the real interest rate (R) and the nominal exchange rate (E) experience shocks with 5% standard deviation. Thus, the variance-covariance matrix of additive noises (Q), will be as follows:[50]

[50] We want the shocks to affect contemporaneous variables only, and not their lagged values. However, if we set to zero the elements of the Q matrix corresponding to lagged variables, Duali will give us an error message. That is why we set those elements equal to the minimum possible value (0.000000001).

$$Q = \begin{bmatrix} 0.0004 & & & & & & & & & & \\ & 0.0025 & & & & & & & & & \\ & & 0.0004 & & & & & & & & \\ & & & 0.0025 & & & & & & & \\ & & & & 0.1^{(-9)} & & & & & & \\ & & & & & 0.1^{(-9)} & & & & & \\ & & & & & & 0.1^{(-9)} & & & & \\ & & & & & & & 0.1^{(-9)} & & & \\ & & & & & & & & 0.1^{(-9)} & & \\ & & & & & & & & & 0.1^{(-9)} & \\ & & & & & & & & & & 0.1^{(-9)} \end{bmatrix}$$

We will perform 100 Monte Carlo runs to compare the performance of the CE procedure against the OLF procedure. To do so, we will use the file `ht-03.dui`. This file is similar to the `ht-02.dui` file used in the previous sections, with some modifications. If we select the Data:Additive Noise Terms:Additive Noise Covariance menu options we will see the Q matrix shown above. In the Specification:Data:Size menu option we have to specify the number of Monte Carlo runs. That option is set to 3. It may be better to make a first run like this with a small number of Monte Carlo runs to gain familiarity with the procedure. However, to perform a more serious experiment, we set it equal to 100. Be aware that this may take some minutes to run, depending on the computer. Then, in the Specification:Source of Random Terms option, we check the options (1) Generate Internally, (2) Uncertain Parameters and (3) System Equations as shown in the dialog box below.

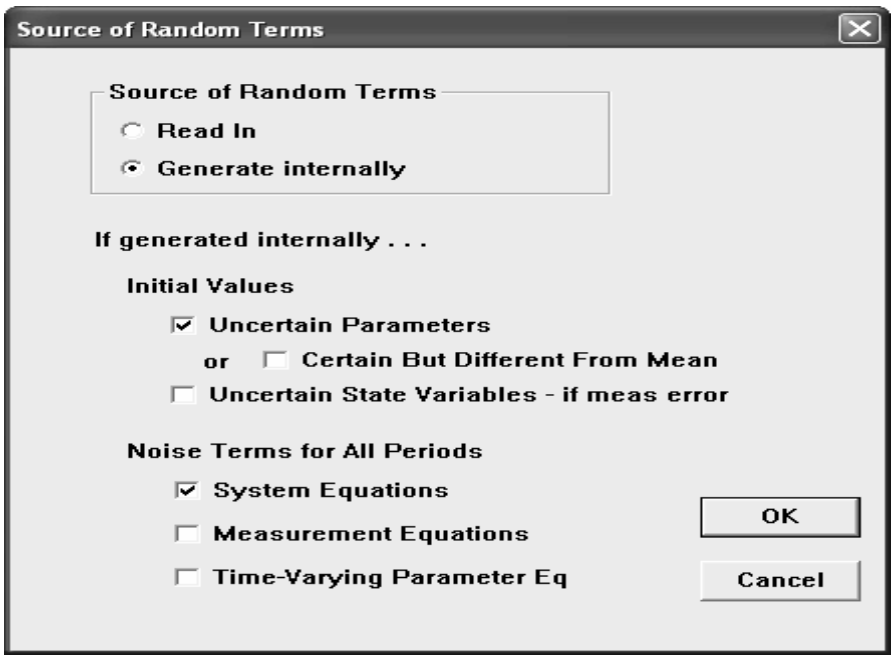


Figure 17.15 Source of Random Terms Dialog Box

We then chose the Solve:Compare Print option, obtaining a dialog box like the one shown below where we see that the options CE and OLF have been selected.

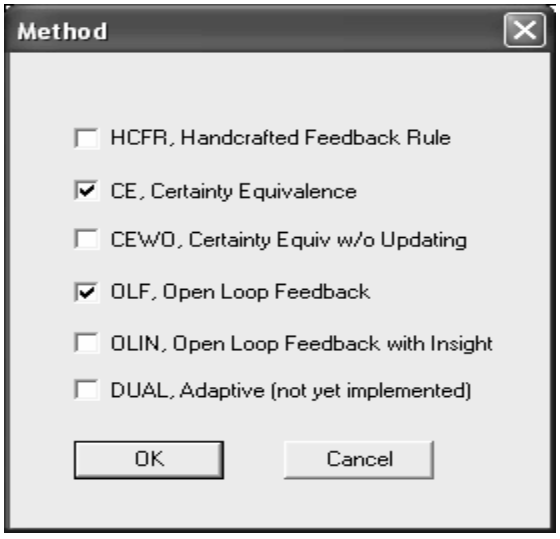


Figure 17.16 Method Dialog Box

When we click OK, we will be asked to provide a debug file name. After doing so, a dialog box like the one shown below will be displayed.

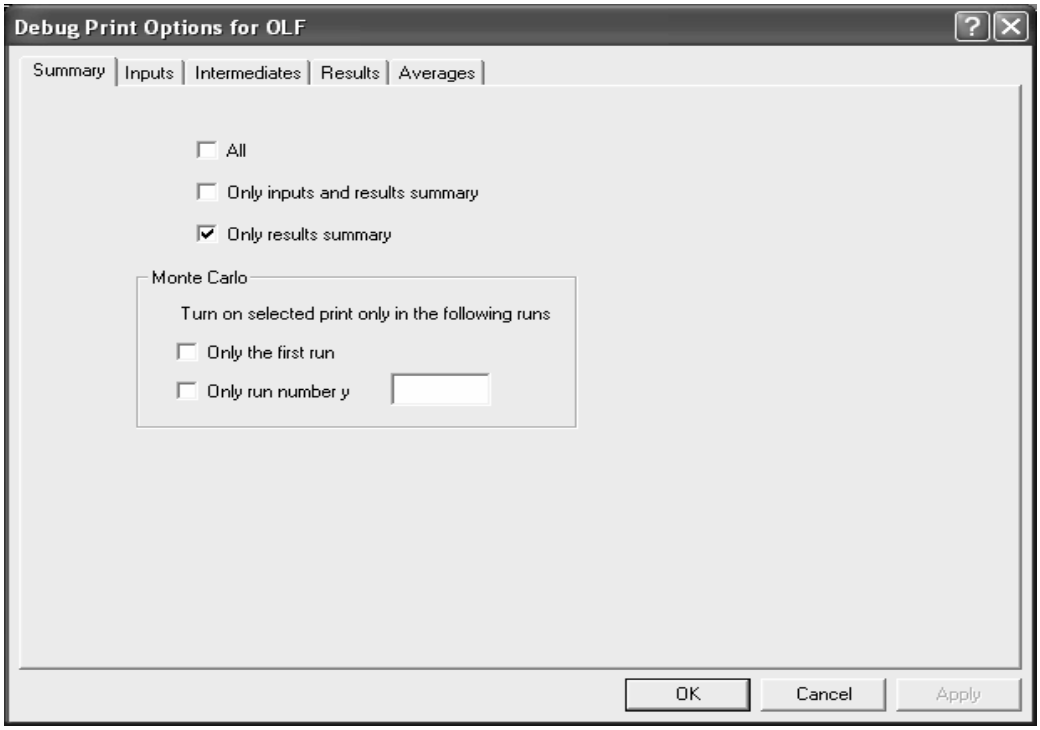


Figure 17.17 Debug Print Options Dialog Box

In this dialog box we have many options to build a very detailed solution report with summary, intermediate and final results, among other things. We just check the "Only results summary" option, leaving all the others blank and then click OK. Duali will start solving the problem. In the meantime, two dialog boxes named Method Count and Average Criterion Values will be displayed. We click OK for each of them. Finally, once the run is completed, the results will be stored in the file we specified as the debug print file. It is best to exit from Duali before examining the results file in an editor. When doing Monte Carlo runs in Duali it is important to look for the results in the debug print file and not in the Display results on line since the Display numbers are only for the last Monte Carlo run and not for the averages across all the runs.

The results in the debug print file corresponding to our 100 Monte Carlo runs are shown in Table 17.2.

	CE	OLF
Average Criterion Value	5.60	5.59
Runs with Lowest Criterion	47	53

Table 17.2 Monte Carlo Results

The Open Loop Feedback procedure does slightly better than the Certainty Equivalence, not only in connection with the average criterion value, but also in terms of the number of Monte Carlo runs with the lowest criterion. As can be appreciated in the graph below, where each diamond represents the value of the criterion function for one Monte Carlo run, most of the diamonds are close to the 45 degree line, indicating a similar performance for both procedures. There are no significant outliers that could be introducing a bias in the computed average criterion values.

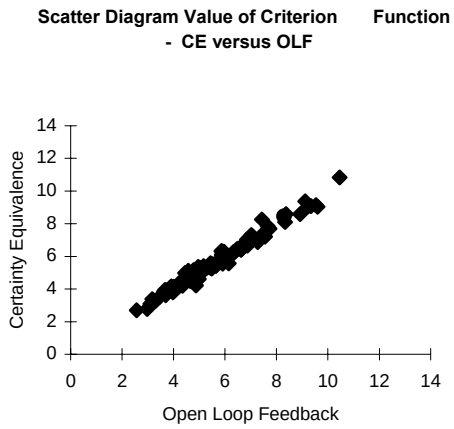


Figure 17.18 Scatter Diagram Value of Criterion Function

These results are against what one would intuitively expect, since in the presence of parameter uncertainty OLF might be expected to do not only slightly but significantly better than CE. However, we have to mention that there are no theoretical results yet developed in connection with the relative performance of CE versus OLF. The experimental results are conditioned on the model structure, its parameter mean and variance values, and may well change (in any direction) in a different context. For

example, working with a different model Amman and Kendrick (1999d) find OLF results that are substantially better than the CE results. Also, Lee (1998) obtains similar results from a substantially larger model.

5. Experiments

In Section 2 of this chapter we analyzed the autonomous and optimal policy responses to a negative shock in net exports. You may want to analyze other shocks implying different initial conditions for the model endogenous variables. Or you may analyze the effects of changes in the exogenous variables and the corresponding optimal policy responses. Also, you may want to put a very high weight (priority) on the money supply or government expenditure so that in fact only one policy variable will be use to control the system. Then, you may contrast these cases against the analysis performed in this chapter in which both controls were assigned equal weights. Finally, you may want to assign different sets of equal values to the weights on the control variables for the experiment presented in this chapter, to observe the displacement effects that these changes have on the optimal policy frontier.

In Section 3 of this chapter we analyzed the optimal response of the policy variables when parameter uncertainty was taken into account. In particular, we increased the relative uncertainty of government expenditure parameters and we found that this induced a more cautious use of that policy variable during the first periods. You may want to continue increasing the level of uncertainty of those parameters and see the pattern of responses in the policy variables. Or you may increase the relative uncertainty of the money supply parameters.

6. Further Reading

For one of the first applications of control theory methods to macroeconomics models see Pindyck (1973). Chow (1975) provides an introduction to the analysis and control of dynamic economic systems. Kendrick (1981) presents a systematic treatment of stochastic control for economic models, with particular focus on passive and active learning methods. Holly and Hughes-Hallett (1989) also present a systematic treatment of optimal control methods, with special treatment of expectations and uncertainty. Sengupta and Fanchon (1997) present methods and a wide range of applications of control theory in economics. Chiarella and Flaschel (2000) provide a nonlinear dynamics

approach to macroeconomics. For a related global dynamics approach to analyzing overlapping generation models see Gomis and Haro (2003).

Kendrick (2005) reviews the historical development and likely future paths in the field of stochastic control in economics.

For a most interesting visual approach to the use of control theory methods in economics that uses the Simulink system with MATLAB see Herbert and Bell (1997). For an observer approach to control methods in economics see Herbert (1998).

Amman and Kendrick (1999a) provide a users' guide to Duali, with a variety of tutorial level chapters dealing with different control methods and models.

Chapter 18

Rational Expectations Macro in Duali

In macroeconomics, the way in which expectations are modeled has a significant effect on model solution and simulation strategies. Some macroeconomic models include the assumption that economic agents form their expectations in a backward-looking adaptive way. That is, in order to form expectations in connection with the likely future value of a given macroeconomic variable, economic agents take into account the recent evolution of that variable, and perhaps of other closely related variables. For example, in the chapter on the Hall and Taylor Model in GAMS, we saw that the expected inflation rate was obtained as a weighted sum of the observed inflation rates in the previous two quarters. From a modeling point of view, that meant that contemporary model expectational variables can be replaced by some combination of lagged variables.

In contrast, the assumption of rational expectations asserts that economic outcomes are not systematically different from economic agents' expectations about those outcomes. This implies that macroeconomic models should embed the notion that economic agents make use of all available information when forming their expectations. Included in agents' information set is thus the model of the economy that the modeler is using to capture their behavior. This assumption has a significant impact in terms of modeling and simulation since under it, agents' expectations are a function of the whole macroeconomic model solution while, at the same time, that solution is a function of agents' expectations. Also, model dynamics becomes more complex, since expectational variables are "forward looking" variables that sometimes will display a "jumping" behavior, instantaneously adjusting to changes in policy or exogenous variables. Finally, policy analysis will also be more demanding, since policymakers will have to take into account the agents' anticipatory behavior to their policy announcements and actions.

In this chapter, we will perform simulations and policy experiments in the Duali software with John Taylor's rational expectations model. This is a prototype one-country model which is very useful as a training ground in the computational modeling of rational expectations. It is also a good introduction to the empirical multicountry models developed by Taylor (1993).

1. John Taylor's Closed Economy Model

John Taylor's closed economy model is a small prototype linear model with staggered contracts and rational expectations variables that generate an interesting pattern of dynamic behavior. It contains the equations, variables and parameters listed below.

Equations

(1.1) $x_t = \frac{\delta}{3}\sum_{i=0}^{2}\hat{w}_{t+i} + \frac{1-\delta}{3}\sum_{i=0}^{2}\hat{p}_{t+i} + \frac{\gamma}{3}\sum_{i=0}^{2}\hat{y}_{t+i}$

(1.2) $w_t = \frac{1}{3}\sum_{i=0}^{2}x_{t-i}$

(1.3) $p_t = \theta w_t$

(1.4) $y_t = -d\, r_t + g_t$

(1.5) $m_t - p_t = -bi_t + ay_t$

(1.6) $r_t = i_t - \hat{p}_{t+1} + p_t$

Variables

x = contract wage
w = average wage
p = price level
y = output
i = nominal interest rate
r = real interest rate
m = money stock
g = government expenditure[51]

where "^ " means expectation through period t.

Parameters

$\delta = 0.5; \quad \gamma = 1; \quad \theta = 1; \quad a = 1; \quad b = 4; \quad d = 1.2.$

The variables (all except i_t and r_t) are logarithms and are deviations from means or secular trends.

---

[51] In the original Taylor model, government expenditure appears implicitly as a shift factor in Eq. (1.4). Here, we make it an explicit variable in that equation.

Eq. (1.1) is a staggered-wage setting equation. It is supposed that a wage decision lasts three years, with one third of the wages being negotiated each year. At any given time t, the contract wage depends on expectations of the values at times t, t+1 and t+2 of wages paid to other workers, the price level and real output. Eq. (1.2) gives the average wage in the economy as the average of the contract wage in the current period and the two previous periods. Eq. (1.3) reflects mark-up pricing behavior by firms, that is, prices are set proportionally to the average wage. Eq. (1.4) defines a standard IS schedule, while Eq. (1.5) is the money demand equation defining an LM schedule. Finally, Eq. (1.6) gives the real interest rate as the nominal interest rate deflated by the rationally expected inflation rate where the expected inflation rate is defined as $\hat{\pi}_t = \hat{p}_{t+1} - p_t$.

The model has 6 equations and 6 endogenous variables. It contains two policy variables: the money stock and government expenditure. The model is dynamic and linear, and has the "natural rate" property, in the sense that nominal shocks may affect real variables in the short-run, but not in the long run.

2. Solving Optimal Control Rational Expectations Problems in Duali

As a rational expectations model, Taylor's model requires specific solution methods different from those applied to standard models. Many methods have been developed over the last two decades for solving rational expectations models. See for example Blanchard and Kahn (1980), Wallis (1980), Fair and Taylor (1983), Anderson and Moore (1985), Oudiz and Sachs (1985), Fisher, Holly and Hughes-Hallett (1986), Pesaran (1987), Juillard (1996), Zadrozny and Chen (1999), Binder and Pesaran (2000), and Sims (2002). Some of those methods are analytical and they usually involve, for the case of linear models, the passage from the model structural form to a "pseudo-reduced form" in which the expectational variables are no longer present. Other methods are numerical. Also, as shown in Holly and Hughes-Hallett (1989) Ch. 7, the analysis of models' dynamic properties such as the computation of eigenvalues and the condition of dynamic controllability become more involved in rational expectation models.

To solve optimal control problems containing rational expectations models, Duali uses a dynamic programming algorithm like the one presented in the Dynamic Optimization chapter, combined with the numerical method developed by Ray Fair and John Taylor to solve rational expectations models. The Fair and Taylor (1983) method is an iterative procedure that starts by solving the model for a set of arbitrary values - usually zeroes - for the path of each forward looking variable. Then, after each iteration,

the values of the forward looking variables are updated with the solution values of the corresponding endogenous variable in the previous iteration. The process stops when convergence is obtained, that is, when the difference between the forward variables values in two successive iterations is smaller than a given tolerance value.

For example, suppose that we have a simple single equation model like the one shown below, in which the future value of a variable (x_{t+1}) is a function of its current value (x_t) and also of its future expected value conditioned on the information available at time t ($x^e_{t+1\,|\,t}$).

$$x_{t+1} = ax_t + bx^e_{t+1\,|\,t}$$

Suppose also that the solution horizon covers only six periods, that $a = 0.4$, $b = 0.1$, and that the initial value for x_t is one. The Excel spreadsheet in Figure 18.1 below shows the results for the first four iterations, where we use E to denote expected value.

Microsoft Excel - MaTayDuali Figure 2.1

File Edit View Insert Format Tools Data Window Help

C9: 1

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Fair-Taylor Rational Expectations Solution Method															
2																
3	a	b														
4	0.4	0.1														
5																
6	period	first iteration				second iteration				third iteration				fourth iteration		
7		X(t+1)	X(t)	E{X(t+1)}		X(t+1)	X(t)	E{X(t+1)}		X(t+1)	X(t)	E{X(t+1)}		X(t+1)	X(t)	E{X(t+1)}
8																
9	1	0.4	1	0		0.44	1	0.4		0.444	1	0.44		0.4444	1	0.444
10	2	0.16	0.4	0		0.192	0.44	0.16		0.1968	0.444	0.192		0.1974	0.4444	0.1968
11	3	0.064	0.16	0		0.0832	0.192	0.064		0.087	0.1968	0.0832		0.0877	0.1974	0.087
12	4	0.0256	0.064	0		0.0358	0.0832	0.0256		0.0384	0.087	0.0358		0.0389	0.0877	0.0384
13	5	0.0102	0.0256	0		0.0154	0.0358	0.0102		0.0169	0.0384	0.0154		0.0173	0.0389	0.0169
14	6	0.0041	0.0102	0		0.0066	0.0154	0.0041		0.0074	0.0169	0.0066		0.0076	0.0173	0.0074
15																
16	example of iteration structure:															
17	second iteration:		cell F1 = a4*g14+b4*h14					cell G14 = F13			cell H14 = B14					
18																

Sheet1 Sheet2 Sheet3

Ready

Figure 18.1: Fair-Taylor Method Example

Notice that four iterations of the model solution are shown in the spreadsheet and that there are three columns of variables shown at each iteration, namely `x(t+1)`, `x(t)` and `E(x(t+1))`. The logic of the procedure is easy to follow. In the first iteration, column D is set to zero. Given those values and the initial value for x_t in cell C9, the

model is solved for each of the remaining five time periods. The results in column B are then copied to column H in the second iteration and the model is solved again. The results are copied from column F to column L and so on. Notice how fast the results converge for this particular model and parameter values - the difference between columns P and N in the fourth iteration is quite small. What makes the Fair and Taylor method attractive is its simplicity, and the fact that it can be applied to multiple equation linear and nonlinear models.

Duali contains a method developed by Amman and Kendrick (1996) to solve optimal control problems with rational expectations. This procedure, which is described below, uses the Fair and Taylor method as an intermediate step.

The problem is expressed as one of finding the controls $(u)_{t=0}^{N}$ to minimize a quadratic "tracking" criterion function *J* of the form:

$$(2.1)\quad J = \left\{ \frac{1}{2}\left[x_N - \tilde{x}_N\right]' W_N \left[x_N - \tilde{x}_N\right] + \frac{1}{2}\sum_{t=0}^{N-1}\left(\left[x_t - \tilde{x}_t\right]' W_t \left[x_t - \tilde{x}_t\right] + \left[u_t - \tilde{u}_t\right]' \Lambda_t \left[u_t - \tilde{u}_t\right] \right) \right\}$$

subject, as a constraint, to the state-space representation of the economic model, also known as the regular form:

$$(2.2)\qquad x_{t+1} = Ax_t + Bu_t + Cz_t + D_1 x^e_{t+1\,|\,t} + D_2 x^e_{t+2\,|\,t}$$

where *x*, *u* and *z* are state, control and exogenous variables respectively, $\tilde{x}$ and $\tilde{u}$ are desired paths for the state and controls variables, and $x^e_{t+1\,|\,t}$ is a "forward looking" variable equal to the expected value of the state variable at period *t+1* conditioned on the information available at time *t*. Also A, B, C, D_1 and D_2 are matrices. In this example, the maximum lead for the forward looking variables is two periods, but it could of course be larger.

A way of formalizing the rational expectations hypothesis is, for a deterministic environment

$$(2.3)\qquad x^e_{t+1\,|\,t} = x_{t+1}$$

and, for a stochastic environment and where E is the mathematical expectation operator

(2.4) $$x^{e}_{t+1\,|\,t} = E_t x_{t+1}$$

Denote the expected value of the state variable at iteration *v* as $x^{ev}_{t+1\,|\,t}$. At the first iteration - iteration zero - the Amman and Kendrick procedure begins by setting $x^{e0}_{t+1\,|\,t} = x^{0}_{t+1} = 0$ for all t, and solving the resulting quadratic linear problem with a standard method such as the one presented earlier in the book in the chapter on Dynamic Optimization. The optimal state variables for the solution obtained - the "no lead" solution - are denoted as x^{NL}. Then, the expected values of the forward looking variables are set equal to the solution for this first iteration, that is:

(2.5) $$x^{e1}_{t+1\,|\,t} = x^{NL}_{t+1} \quad \text{and} \quad x^{e1}_{t+2\,|\,t} = x^{NL}_{t+2} \quad \text{for all } t\,.$$

Thus, the system of equations corresponding to the first iteration is now:

(2.6) $$x^{1}_{t+1} = Ax^{1}_{t} + Bu^{1}_{t} + Cz_t + D_1 x^{e1}_{t+1\,|\,t} + D_2 x^{e1}_{t+2\,|\,t}$$

Notice that the terms:

(2.7) $$Cz_t + D_1 x^{e1}_{t+1\,|\,t} + D_2 x^{e1}_{t+2\,|\,t}$$

are all known. This allows us to write the system of equations as:

(2.8) $$x^{1}_{t+1} = Ax^{1}_{t} + Bu^{1}_{t} + \tilde{C}\tilde{z}^{1}_{t}$$

where:

(2.9) $$\tilde{C} = \begin{bmatrix} C & D_1 & D_2 \end{bmatrix} \quad \text{and} \quad \tilde{z}^{1}_{t} = \begin{bmatrix} z_t \\ x^{e1}_{t+1\,|\,t} \\ x^{e1}_{t+2\,|\,t} \end{bmatrix}.$$

Again, we have a quadratic linear problem which can be solved with standard methods. Once we do so, we will have another set of solution values for the state variables which will be used as the values of the forward-looking variables in the next iteration, and so on. The procedure will stop when convergence is obtained.

3. The Taylor Model in Duali

In the following we will focus on the implementation of Taylor's model in Duali to perform simulations and optimal policy analysis. First we will transform the model equations to make them more suitable for a matrix representation. As presented in Section 1, the model was:

(1.1) $x_t = \frac{\delta}{3}\sum_{i=0}^{2}\hat{w}_{t+i} + \frac{1-\delta}{3}\sum_{i=0}^{2}\hat{p}_{t+i} + \frac{\gamma}{3}\sum_{i=0}^{2}\hat{y}_{t+i}$

(1.2) $w_t = \frac{1}{3}\sum_{i=0}^{2} x_{t-i}$

(1.3) $p_t = \theta w_t$

(1.4) $y_t = -d\, r_t + g_t$

(1.5) $m_t - p_t = -bi_t + a y_t$

(1.6) $r_t = i_t - \hat{p}_{t+1} + p_t$

Expanding the summation signs, renaming some variables, and substituting the corresponding numerical values for the model parameters, we obtain the model below.

(3.1) $x_t^{cw} = 0.1\bar{6} w_t + 0.1\bar{6} p_t + 0.\bar{3} y_t + 0.1\bar{6}\hat{w}_{t+1} + 0.1\bar{6}\hat{p}_{t+1} + 0.\bar{3}\hat{y}_{t+1} + 0.1\bar{6}\hat{w}_{t+2} + 0.1\bar{6}\hat{p}_{t+2} + 0.\bar{3}\hat{y}_{t+2}$

(3.2) $w_t = 0.\bar{3} x_t^{cw} + 0.\bar{3} xl_t^{cw} + 0.\bar{3} xl_{t-1}^{cw}$

(3.3) $p_t = w_t$

(3.4) $y_t = -1.2 r_t + g_{t-1}$

(3.5) $i_t = 0.25 y_t + 0.25 p_t - 0.25 m_{t-1}$

(3.6) $r_t = i_t + p_t - \hat{p}_{t+1}$

(3.7) $xl_t^{cw} = x_{t-1}^{cw}$

Notice that x_t^{cw} is the contract wage in Taylor's model, which we re-labeled here in Eqs. (3.1) and (3.2) to avoid notational confusion with x_t, which will be the vector of stacked variables of the model matrix representation. Also notice that since in Taylor's model expectations are conditional on the information available at time t, we can write:

(3.8) $w_{t|t}^e = w_t, \quad p_{t|t}^e = p_t, \quad y_{t|t}^e = y_t$

This is why the variables in the first three right-hand side terms in Eq. (3.1) are actual values and not expected values, as is the case in the remaining terms.

In Eq. (3.2) there is a new variable xl_t^{cw} which is defined in Eq. (3.7) as equal to lagged x_t^{cw}, that is, x_{t-1}^{cw}. Therefore, the variable xl_{t-1}^{cw} in Eq. (3.2) will be equal to x_{t-2}^{cw}. In this way, using the same method we employed in the Hall and Taylor in Duali chapter, we produce a one-lag-order reduction of Eq. (1.2). Since this is the only lagged equation in the model, we are left with a first order model representation suitable to be used in optimal control experiments.

In Eq. (3.5) we moved the interest rate *i* to the left-hand side to make its role as a state variable explicit. Finally, in Taylor's model, *m* and *g* appear as contemporaneous to the endogenous variables. By assuming that there is a one period lag between a policy decision and its implementation, we can redefine these two control variables in Eqs. (3.4) and (3.5) as m_{t-1} and g_{t-1}, since Duali, as well as the optimal control literature, works with one-lag policy variables.

We will now represent the model in what is known as the Pindyck or "I-A" form, which is an equivalent representation to the form presented in Eq. (2.2). The Pindyck form of Taylor's model can be written as shown below in Eq. (3.9).[52]

$$x_t = A_0 x_t + A_1 x_{t-1} + B_1 u_{t-1} + C_1 z_{t-1} + \hat{D}_1 x^e_{t/t} + \hat{D}_2 x^e_{t+1/t} + \hat{D}_3 x^e_{t+2/t} \tag{3.9}$$

where

$$x_t = \begin{bmatrix} x_t^{cw} \\ w_t \\ p_t \\ y_t \\ i_t \\ r_t \\ xl_t^{cw} \end{bmatrix}, \quad u_{t-1} = \begin{bmatrix} m_{t-1} \\ g_{t-1} \end{bmatrix}, \tag{3.10}$$

[52] In Taylor's model, expectations are conditioned on the information available at "t". In Duali, when a model is written in the Pindyck form, expectations are conditioned at "t-1". This change in the timing of the information will not appear as problematic for the Taylor model, since Duali will replicate the results obtained by the original Taylor simulations. However, different assumptions concerning the information set timing may be relevant for other models.

$$A_0 = \begin{bmatrix} 0 & 0.1666 & 0.1666 & 0.3333 & 0 & 0 & 0 \\ 0.3333 & 0 & 0 & 0 & 0 & 0 & 0.3333 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1.2 & 0 \\ 0 & 0 & 0.25 & 0.25 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$A_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.3333 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad B_1 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ -0.25 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\hat{D}_2 = \begin{bmatrix} 0 & 0.1666 & 0.1666 & 0.3333 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\hat{D}_3 = \begin{bmatrix} 0 & 0.1666 & 0.1666 & 0.3333 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

In Eq. (3.9), z_{t-1} is a vector of exogenous variables, while C_1 is a matrix. They are both equal to zero, since the model does not contain exogenous variables. Notice also that $\hat{D}_1$ is set equal to zero, since the model does not contain contemporaneous expected

variables. Finally, we set equal to four the maximum number of decimals for parameter values.

4. Dynamic Simulation

As a way of getting acquainted with some dynamic properties of the Taylor's model, we will analyze the dynamic evolution of the model for given changes in its policy variables. The general problem to be solved in Duali is the one of finding the controls $(u)_{t=0}^{N-1}$ to minimize a quadratic "tracking" criterion function J of the form

$$(2.1)\quad J = \left\{ \frac{1}{2}\left[x_N - \tilde{x}_N\right]' W_N \left[x_N - \tilde{x}_N\right] + \frac{1}{2}\sum_{t=0}^{N-1}\left(\left[x_t - \tilde{x}_t\right]' W_t \left[x_t - \tilde{x}_t\right] + \left[u_t - \tilde{u}_t\right]' \Lambda_t \left[u_t - \tilde{u}_t\right] \right) \right\}$$

subject to:

$$(3.9)\qquad x_t = A_0 x_t + A_1 x_{t-1} + B_1 u_{t-1} + C_1 z_{t-1} + \hat{D}_1 x^e_{t/t} + \hat{D}_2 x^e_{t+1/t} + \hat{D}_3 x^e_{t+2/t}$$

where variables and parameters were defined in the previous sections.

Though the Duali software is oriented toward solving optimization problems like the one just presented, it can also handle standard simulations like the experiments to be performed in this section where we will change the values of the policy variables to see their dynamic impacts on the endogenous variables of the model. To do so, the weights on the controls in the Λ matrix are set to relatively high values, while the weights on the states in the W matrix are set to relatively small values. Then we define the desired paths for the controls as equal to the policy change to be introduced. In this way we force the system to respond to the pre-specified changes in the policy variables. In fact, what we are doing is ignoring the optimization part of the solution method presented in the previous section and using the Fair-Taylor method only to simulate the rational expectations model.

We begin from the main menu shown in the Duali main window in Fig 2 below.

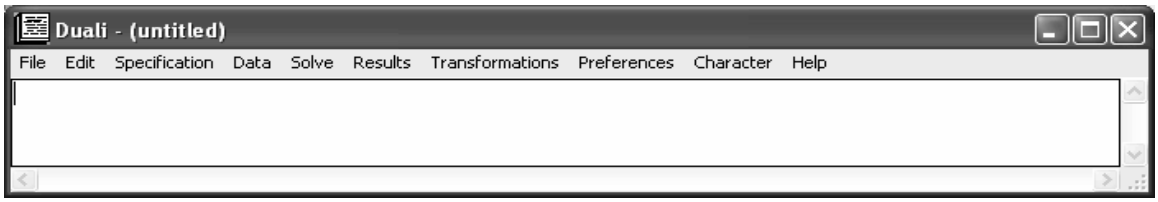


Figure 18.2 Duali Main Window

From the File option we open the file `tay-sim.dui`. In the Specification:Stochastic Terms menu option, we see that the problem is set as deterministic, as shown below.

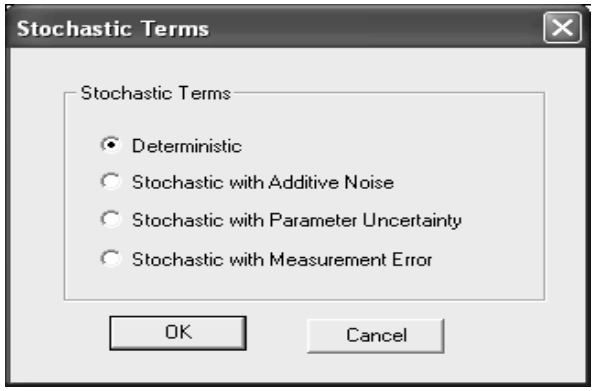


Figure 18.3 Stochastic Terms Dialog Box

We then select the Specification:Functional Form option and we obtain the dialog box shown below.

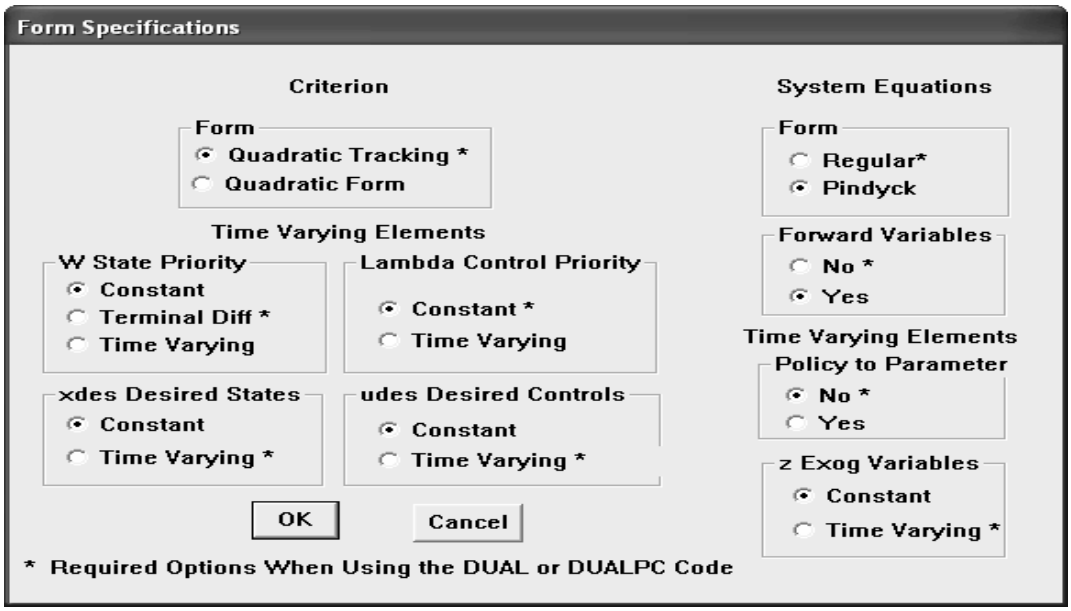


Figure 18.4 Form Specification Dialog Box

On the Criterion side of the dialog box we see that the problem is a Quadratic-Tracking problem with constant state and control priorities. Also the desired states and controls are constant. They will all be set equal to zero, since later in the optimal control experiment in the next section we will seek to minimize deviations of target variables from means or secular trends. Also recall that the model variables are already expressed in deviation form.

On the System Equations side of the dialog box in Fig. 18.4 we see that the Pindyck form is selected, while the option Yes is also selected for Forward Variables, there are no policy to parameter effects and the exogenous variables are constant.

From the Data:Size menu we obtain the dialog box below.

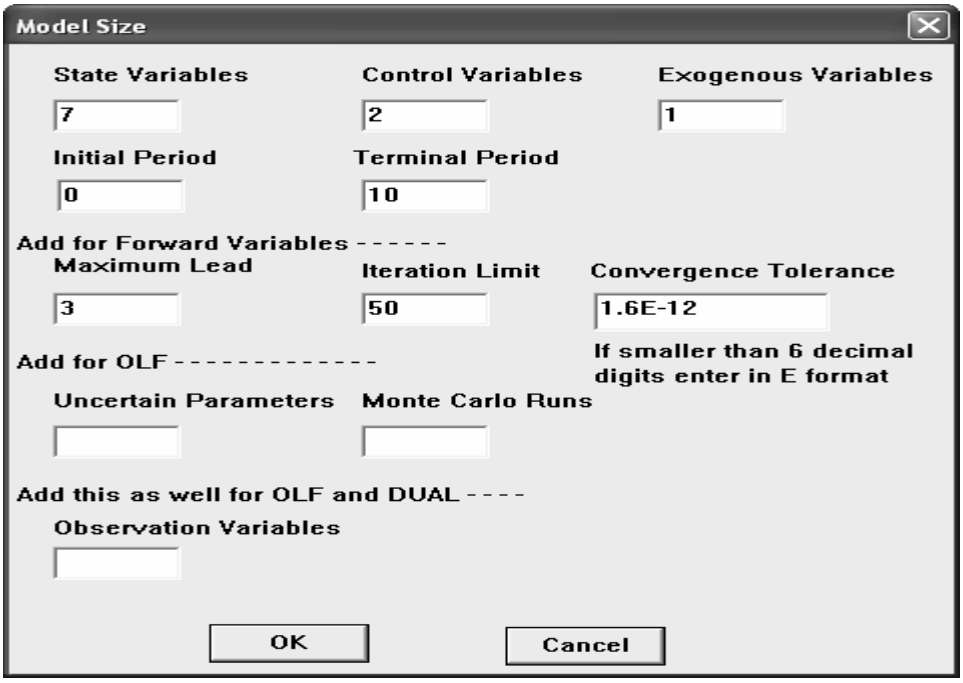


Figure 18.5 Model Size Dialog Box

The model is specified as containing seven state variables (in fact, six contemporaneous and one lagged), two control variables and one exogenous variable, and the simulation covers eleven periods. The Maximum Lead for forward variables is set as equal to three. This is telling Duali that the model contains three $\hat{D}$ matrices, as seen in Eq. (3.9) above. The Iteration Limit is set to 50 and the Convergence Tolerance to 1.6E-12, that is, a very small number in exponential notation. Thus, if the sum of squared difference between all the control variables in all time periods in one iteration and the

previous iteration is less than the convergence tolerance number, then the iterations are halted and convergence is declared. Otherwise, if convergence is not achieved once the iteration limit is reached, an error message is displayed.

Such a small convergence tolerance number will be necessary to perform simulation experiments in which we will force the controls to follow given paths, thus allowing them to experience very minor changes from period to period. Therefore, since Duali computes convergence over changes in the controls, and given that we will allow only minor changes in them, we need to impose a very small convergence tolerance number to be able to run simulation experiments. Such a small number will not be necessary in the optimal policy experiments to be introduced later in this chapter.

The Data:Acronyms menu option contains the assignment of labels to the model variables and time periods. The Data:System Equations section contains the numerical values for matrices A_0, A_1, B_1, C_1, $\hat{D}_1$, $\hat{D}_2$ and $\hat{D}_3$. The Data:Criterion section contains the values for the W and Λ weighting matrices and the desired paths values for state and control variables. We see that the weights on the controls in the Λ matrix are set to 99, a relatively large value, while the weights on the states in the W matrix are set 1, a relatively small number, and the desired paths for the controls are set equal to the policy change to be introduced, i.e., to 0.01 for the experiment to follow. In this way we force the system to respond to the pre-specified changes in the policy variables. Finally, choosing the menu option Solve:QLP the problem is solved and the numerical results are automatically displayed. The menu option Results allows us to define different display, plotting and printing options. Also, for this and the other experiments in this chapter, it will be convenient to set the display of results to four decimals. This can be done in the Preferences:Results menu option, choosing the corresponding value in the Format section.

Figure 18.6 below show the results of two experiments: a 1% unanticipated permanent increase in the money supply (*m*) and a 1% unanticipated permanent increase in government expenditure (*g*). That is, *m* and *g* increase by 0.01 at the first period of each of the two experiments, and are kept at their new value from the second period onwards. On the horizontal axes are the time periods. For *y* and *p*, the vertical axes correspond to percent deviations from steady-state values, while for *i* and *r* the vertical axes show percent points.[53] Thus a value of 0.01 in the GDP graph means that GDP goes

[53] Remember that in Taylor's model, y and p are in logs, which is equivalent to percent deviations from steady-state while i and r are not.

from 600 to 606 billion dollars, while a value of 0.01 in the nominal interest rate graph means that that rate changes from 5% to 6%.[54]

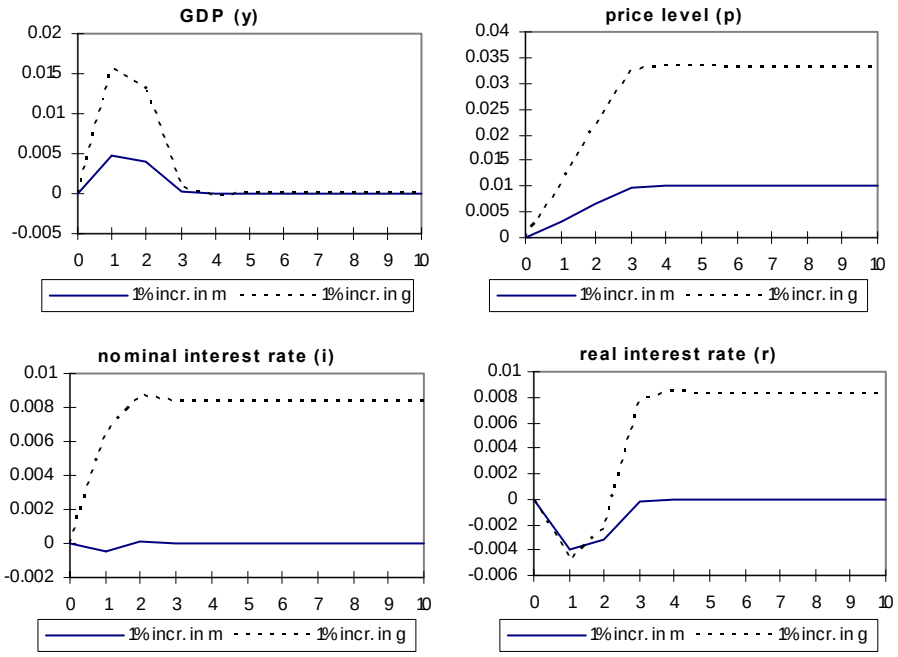


Figure 18.6 Dynamic Simulations of Changes in Policy Variables

Here is how John Taylor explains the observed behavior of the model for the two experiments:

"Monetary policy has an expected positive effect on output that dies out as prices rise and real-money balances fall back to where they were at the start. Note that the real interest rate drops more than the nominal rate because of the increase in expected inflation that occurs at the time of the monetary stimulus. For this set of parameters the nominal interest hardly drops at all; all the effect of monetary policy shows up in the real interest rate. Fiscal policy creates a similar dynamic pattern for real output and for the price level. Note, however, that there is a surprising "crowding-in" effect of fiscal policy in the short run as the increase in the expectation of inflation causes a drop in the real interest rate. Eventually the expected rate of inflation declines and the real interest rate

[54] Taylor (1993), Chapter 1, present graphs conveying the same information as the ones we show here. However, he presents the results in levels.

rises; in the long run, private spending in completely crowded out by government spending."[55]

5. Optimal Policy Analysis

We will now apply optimal control techniques to Taylor's model. The problem is to find the optimal paths for the policy variables given desired paths for the target variables, and it can be stated in the same form as was done before at the beginning of section 4. We will assume that the policy goal is to stabilize y, p, i and r around steady-state values (that is, around zero). We will put high and equal weights on stabilizing y and p, lower and equal weights on i and r, and even lower weights on the policy variables m and g. The corresponding weighting matrices, shown below, will remain constant through time.

(5.1)
$$W = \begin{bmatrix} 0 & & & & & & \\ & 0 & & & & & \\ & & 100 & & & & \\ & & & 100 & & & \\ & & & & 50 & & \\ & & & & & 50 & \\ & & & & & & 0 \end{bmatrix}, \quad \Lambda = \begin{bmatrix} 25 & \\ & 25 \end{bmatrix}$$

To perform a deterministic experiment, we will assume that the economy is going through a recession provoked by a temporary adverse shock to y which brings it 4% below its steady-state value. What would be, in this situation, the optimal paths for m and g? What would be the optimal path for the state variables as compared with the autonomous response of the system?

To perform this experiment in Duali, we use the file `tay-qlp.dui` that is essentially the same as the one used in the previous section, with some modifications. In the Data:Criterion section we see that the values of the W and Λ weighting matrices are now set as in Eq. (5.1), while the desired paths for the controls are set to zero.

In order to implement the shock to y in the first simulation period we have to introduce an artificial time-varying exogenous variable. That is, the shock will be

[55] Taylor (1993), page 25.

defined as a first-period change in an arbitrary exogenous variable affecting the state variable *y* only. To do so, in the Specifications:Functional Forms option, in the "z Exog Variables" section the option Time Varying is now selected, as shown in the dialog box below.

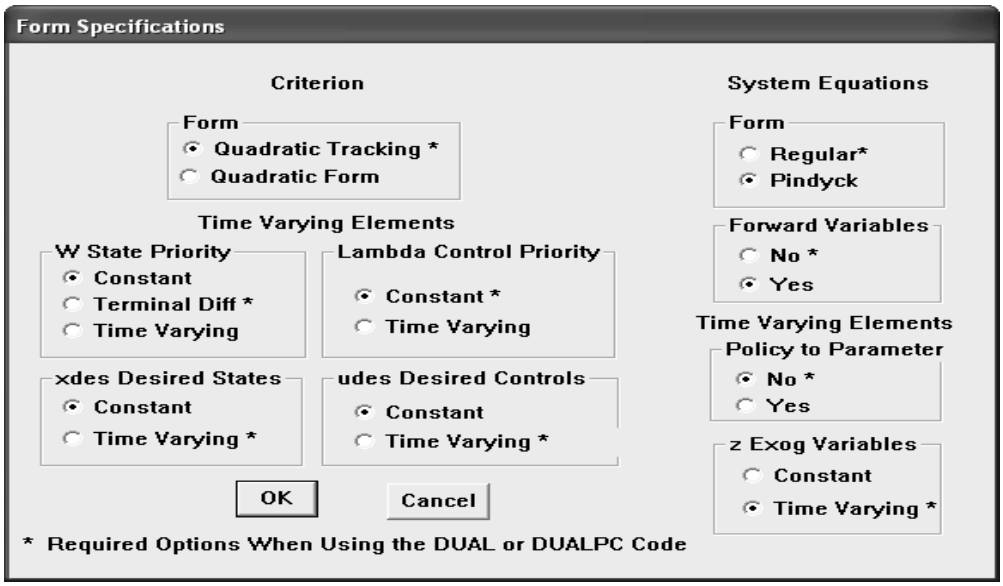


Figure 18.7 Form Specifications Dialog Box

Then, in the Data:System Equations option we set the fourth element of the matrix C_1 equal to 1 and set the first element of the exogenous variable *z* equal to -0.04 while all the remaining elements are set to zero, as shown in the dialog boxes below.

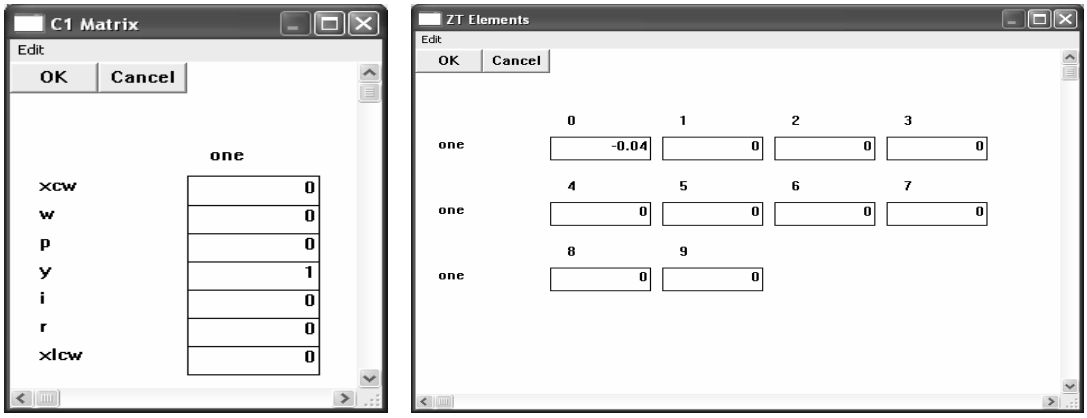


Figure 18.8 C1 Matrix and ZT Elements Input Windows

Notice also that this procedure is different from the one we used to implement an analogous shock in the Hall and Taylor in Duali chapter. There, we applied the shock to the initial value of the shocked variable, that is, we defined the shock in the Duali option Data:System Equations:x0. We cannot do that here, since the variable of interest (*y*) does not appear with lagged values in Taylor's model.[56]

Finally, we solve the problem choosing the menu option Solve:QLP. The graphs in Fig. 18.9 below show the autonomous response of the system to a -0.04 unanticipated transitory shock to *y*, and the behavior obtained when applying deterministic optimal control (QLP) to face the same shock, that is, when actively using *m* and *g* as controls.

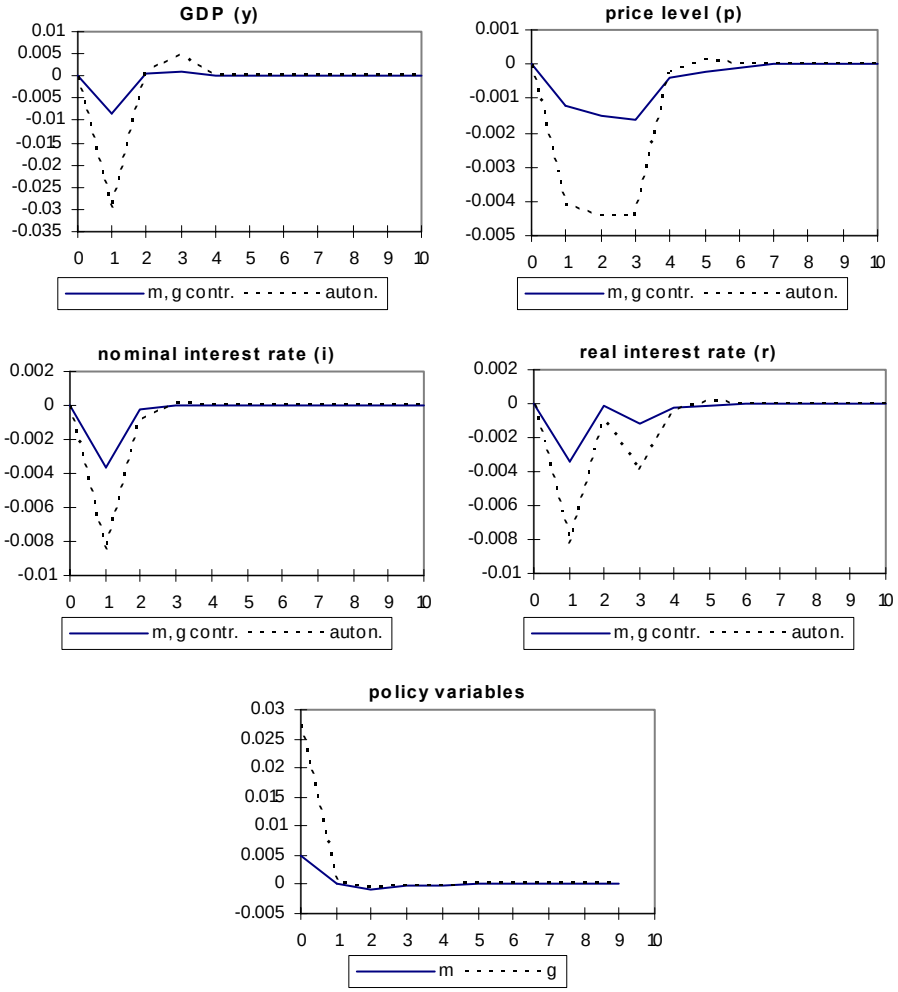


Figure 18.9 Autonomous Response vs. Optimal Policy Experiment

[56] We could use the option System Equations-x0 if, instead of shocking the variable "y", we decide to shock the contract wage, since the contract wage is the only variable with lagged values in Taylor's model.

We can observe how the behavior of the state variables under the optimal control solution outperforms substantially the autonomous response of the system, reducing the costs of getting the economy out of the recession. In order to generate that behavior, as can be seen in the policy variables graph, the optimal policy mix relies on a 2.5% transitory expansion in government expenditure during the first period, at the same time that is also requires a small 0.5% transitory increase of the money supply during the first period.

It may be surprising to find such a positive active policy role in the presence of rational expectations since that specification is sometimes identified with the idea of policy ineffectiveness. However, we have to remember that Taylor's model contains a built-in rigidity - a staggered contracts mechanism - that breaks down the ineffectiveness of policy in the short-run.[57]

More generally, rational expectations will tend to increase the degree of controllability of an economic system, unless the particular structure and/or parameter values of the model imply a complete neutralization of the policy variables effects.[58] Indeed, not only can the policy-maker influence the economy through past and current controls, but he can also affect the economic system through the pre-announcement of future control values. However, for these announcements to have a positive effect on the economic performance, they have to be credible, that is, the policy-maker has to be committed to carry them out.[59] These issues have led some researchers to focus their policy analysis on the evaluation of alternative rules that policymakers are presumed to follow. Two of the most influential researchers engaged in this type of work are John Taylor and Michael Woodford.[60]

For example, using the Taylor model, we may be interested in evaluating the performance of a monetary policy rule in which the monetary authority, having as an implicit target the stabilization of the price level, changes the money stock in an inverse

---

[57] To learn about the role of nominal and real "rigidities" in macroeconomic models, see Blanchard and Fischer (1989).

[58] See Holly and Hughes-Hallett (1989), Chapter 7.

[59] Lack of credibility may lead to problems of "time inconsistency". See Holly and Hughes-Hallett (1989), Chapter 8; and Blanchard and Fischer (1989), Chapter 11. For an appraisal of the practical importance of this issue, see Blinder (1997).

[60] See Taylor (1998) and Woodford (2003).

proportion to the changes in the price level. In formal terms, a simple rule of that type can be written as

(5.2) $$m_t = a_r p_t$$

where *m* is the money stock, p is the price level, and a_r is a negative constant coefficient which in control theory is called the feedback gain coefficient. Our goal will be to evaluate how the variance of the price level changes as the absolute value of the a_r coefficient increases, that is, as the monetary authority responds more strongly to changes in the price level, when the model is shocked by an additive noise.

To perform these experiments in Duali we use the file `tay-hcfr.dui`. In the Specification:Stochastic Tems option, the problem is defined as stochastic with additive noise, as shown in the dialog box below.

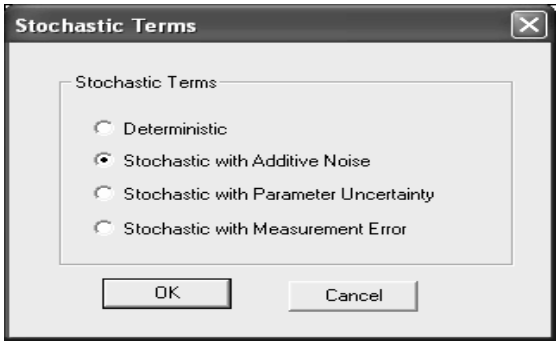


Figure 18.10 Stochastic Terms Dialog Box

In a similar fashion as in the Quadratic Linear Problem above, in the Specifications:Functional Forms option, in the "z Exog Variables" section the option Time Varying is selected. But here we have to do so in order to be able to define the source of random terms. Then, in the Specification:Source of Random Terms option, the Generate Internally option is selected as shown in the dialog box below, indicating that Duali's random numbers generator will be used to generate the shocks. Also, in the Noise Terms for All Periods section, the System Equations option is selected, indicating that the shocks will be applied to the system equations only.

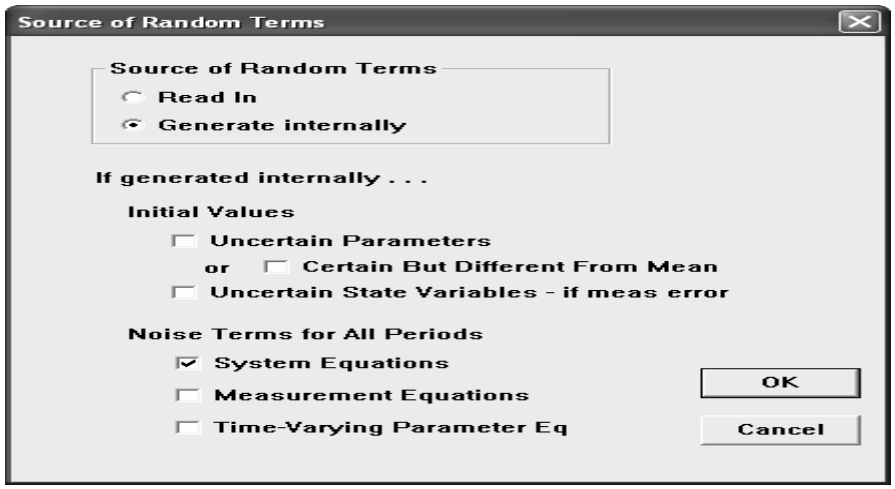


Figure 18.11 Sources of Random Terms Dialog Box

In the Specification:Options Monte Carlo option, we can select the starting period for the calculation of the variance of state and control variables over time. As shown in the dialog box below, we selected period zero as the starting period.

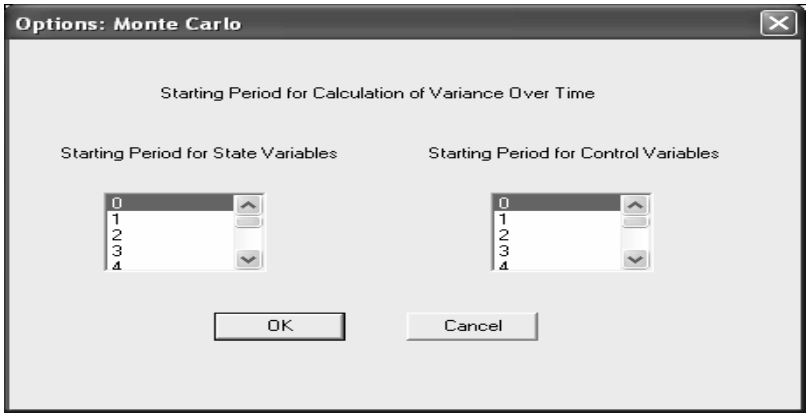


Figure 18.12 Monte Carlo Options Dialog Box

In the Data:Size option, we now have to specify the number of Monte Carlo runs. As shown in the dialog box below, we chose 1000.

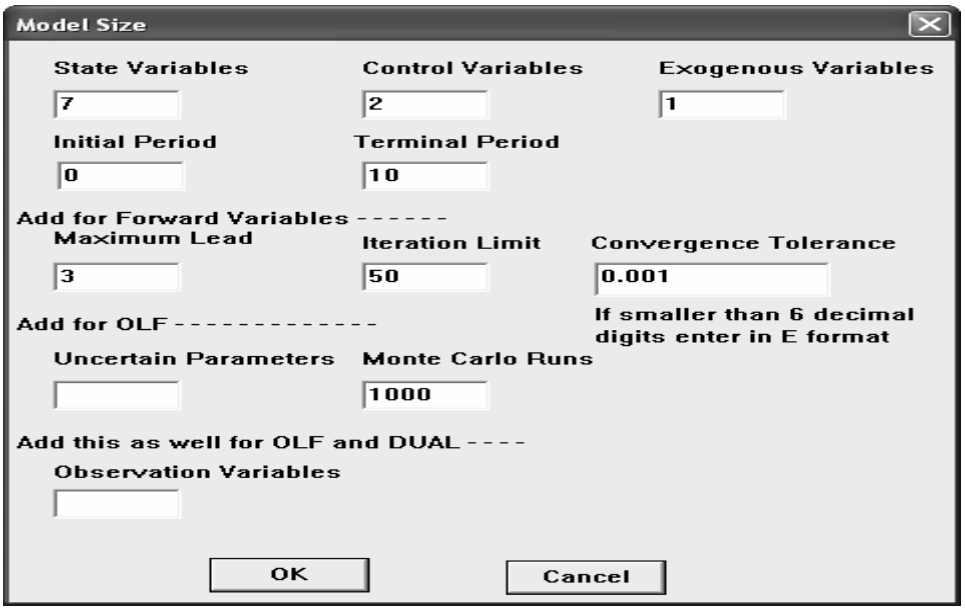


Figure 18.13 Model Size Dialog Box

The next step is to define the variance of the shocks to be applied to the model during the Monte Carlo runs. We will perform experiments in which the shocks will be applied to the contract wage equation only. To do so, as shown in the dialog boxes below, we first select, in the Data:Additive Noise Terms option, the Q, Additive Noise Covariance option. This selection will cause the display of the Q covariance matrix of the additive noise terms. There we assign the value 0.1 to the diagonal element corresponding to the contract wage variable x^{cw}. All the other values should be zeroes. However, having zeroes in the diagonal of the Q matrix will cause problems when Duali tries to find its inverse during the solution of the problem. Thus, we assign very small values (0.00001) to the remaining diagonal elements.

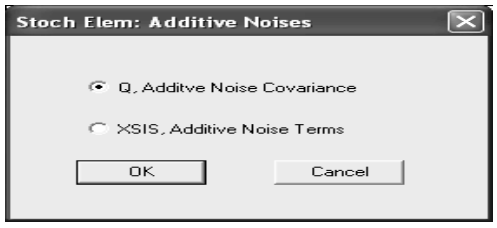


Figure 18.14 Additive Noise Terms Dialog Box

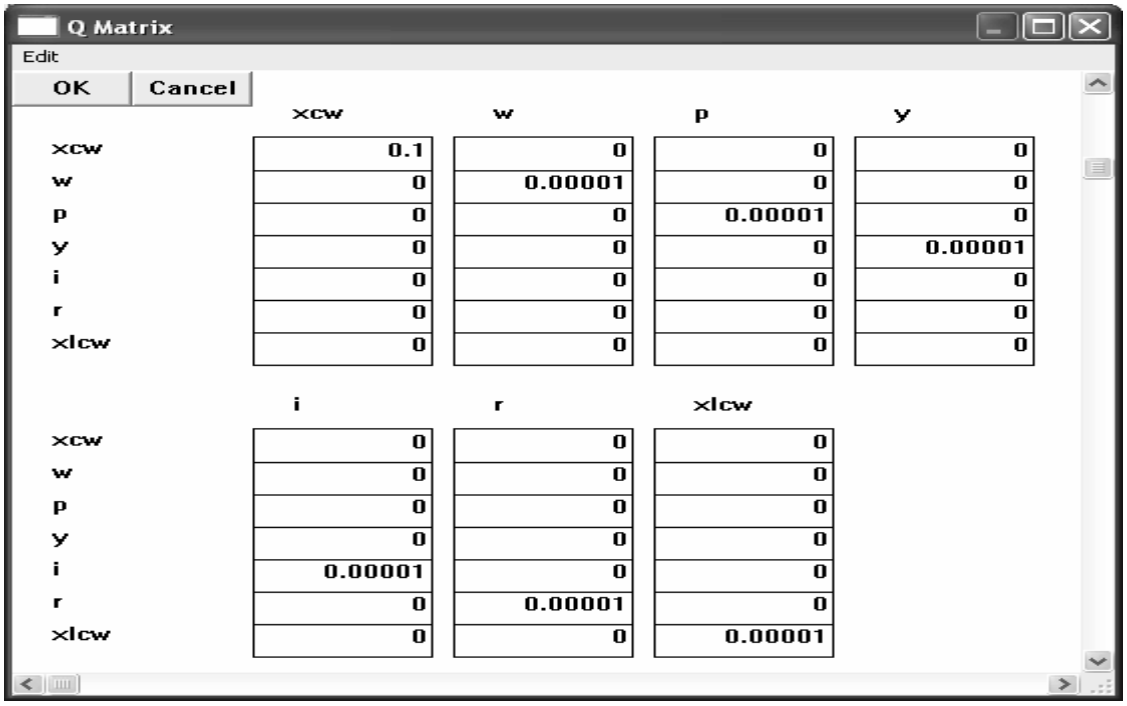


Figure 18.15 Q Matrix Input Window

Having defined the stochastic structure of the simulations, we now have to define and assign values to the feedback rule to be evaluated. The mathematical form of this rule was defined earlier in Eq. (11) of the Dynamic Optimization chapter as

$$u_k = G_k x_k + g_k$$

Thus to modify the feedback rule we need to change the elements in either the feedback gain matrix G or the feedback gain vector g. To do so, we select the Data:Handcrafted Feedback Rule option. As shown in the dialog box below, we select the capital G option, which is the feedback gain matrix of the rule to be applied in the experiments. We leave the small g option blank, since it corresponds to a vector of constant terms that are absent from the specific rule we will evaluate as defined in Eq. (5.2) above, i.e.

(5.2) $$m_t = a_r p_t$$

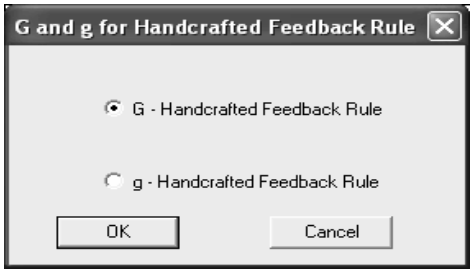


Figure 18.16 G and g Dialog Box

When making the selection of G, the corresponding window will be displayed as shown below. We see that the value -0.1 is the only one assigned. It corresponds to the value of the a_r coefficient in Eq. (5.2). We also see that more complex rules could be easily defined by assigning values to other cells in the matrix.

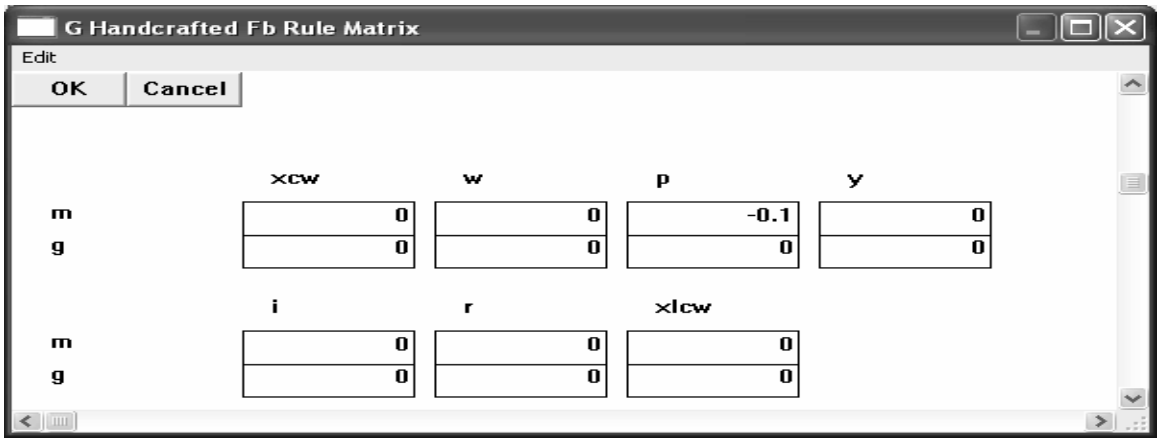


Figure 18.17 Feedback Gain Matrix G Input Window

Having defined the stochastic structure of the experiments to be performed, and the rule to be evaluated, we are now ready to move on to the selection of the solution method and the storage and display of results. We first select the Solve:Compare Print option. We will obtain a dialog box which displays several solution methods. We could select some or all of them in case we want to perform experiments comparing their relative performance. Since that is not our goal here, we just select the HFCR, Handcrafted Feedback Rule option as shown in the dialog box below.

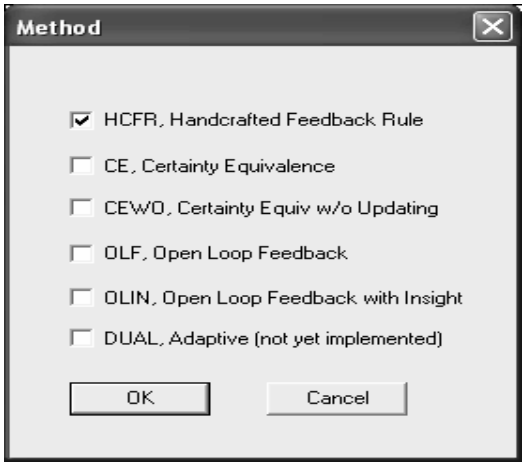


Figure 18.18 Method Dialog Box

When doing so, we will be asked to provide a debug file name, for example we could use the name `tay-hcfr.dbg`. This file will contain the simulation results. After providing the file name, a dialog box containing many options related to the generation of results will be displayed, as shown in the dialog box below. Given the nature of our experiment, we will keep all the options blank except two. In the Averages section, we will select the Average Average over Monte Carlo Runs and the Average Variance over Monte Carlo Runs options.

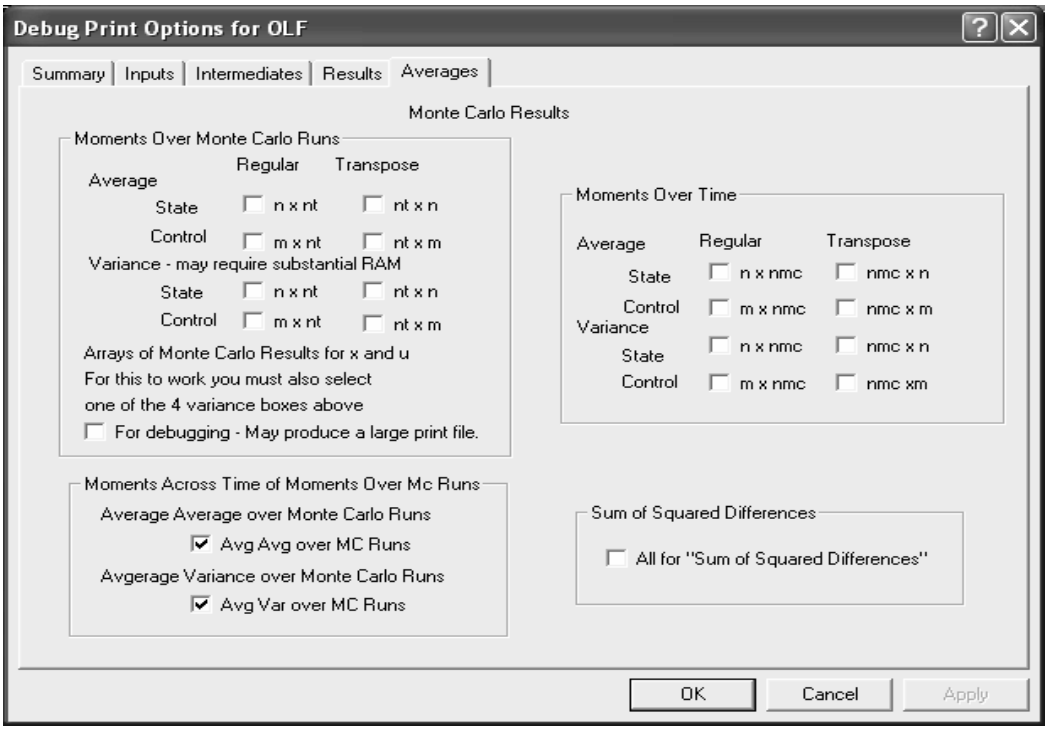


Figure 18.19 Debug Print Dialog Box

We are then ready to perform our experiment. Once we click the OK button, the Monte Carlo runs will begin. Since we are performing 1000 runs, it may take a while before results are displayed. Two dialog boxes like the ones shown below will appear while Duali is running (one after the other). We should just dismiss them by clicking the OK button, since they display results corresponding to experiments with cross comparison of methods, something we are not interested in here.

Figure 18.20 Method Count and Average Criterion Value Windows

We exit from Duali and then open the debug file, `tay-hcfr.dbg` as we named it above, with an editor. Since we have performed a large number of Monte Carlo runs, the output will be quite large since it will display some basic results corresponding to each run. Moving down to the end of the output, our results of interest are just the following ones:

```
AvgVarXsTimeHcfr
      0.0924    0.0168    0.0168    0.0129    0.0003
AvgVarXsTimeHcfr
      0.0089    0.0825
....
AvgVarUsTimeHcfr
      0.0002    0.0000
```

AvgVarXsTimeHcfr means the average variance of the state variables across time for the handcrafted feedback rule solution method. We see that there are seven results, each one corresponding to an element of the transpose of the state variable vector as defined in Eq. (3.10) above that is:

(5.3) $$x_t^{'} = [x_t^{cw} \quad w_t \quad p_t \quad y_t \quad i_t \quad r_t \quad xl_t^{cw}]$$

Thus, our result of interest is the third one to the right in the first row (0.0168) since it corresponds to the average variance of the price level variable. AvgVarUsTimeHcfr contains the results corresponding to the control variables, so the first one (0.0002) the one corresponding to the variance of the lagged money supply stock is the one of interest to us.

These results considered by themselves are not very informative. However, we can repeat the experiment for different values of the a_r coefficient to obtain a comparative performance. Table 18.1 below shows the results of ten experiments.

a_r	Variance of p	Variance of m
-0.1	0.0168	0.0002
-0.2	0.0156	0.0006
-0.3	0.0158	0.0014
-0.4	0.0157	0.0024
-0.5	0.0155	0.0038
-0.6	0.0150	0.0053
-0.7	0.0154	0.0072
-0.8	0.0149	0.0093
-0.9	0.0145	0.0113
-1	0.0143	0.0138

Table 18.1 Comparative Rules Experiments

We can see that as the absolute value of the feedback gain coefficient a_r increases, the variance of the price level tends to decrease, while the variance of the money stock increases. That is, a stronger response of the monetary authority to changes in the price level reduces the variance of that variable but at the cost of an increased variance of the policy tool. A natural question to be asked is what would be the optimal rule, in this case, i.e. the optimal level of the feedback gain coefficient. If the only concern is the variance of the price level, the response is easy: it will be the highest possible absolute value. However, if the variance of the money stock is also a concern, relative priorities should be explicit.

6. Experiments

As a first and relatively simple experiment, you can perform optimal policy experiments like the one presented in Figure 18.6, changing the priorities on state and control variables. Then, you can also change the nature of the initial shock.

Alternatively you can specify different handcrafted feedback rules to perform experiments like the one presented in Figure 18.9. For example, you may want to specify a rule in which the money supply is a function of output instead of the price level. Or you may want to design more complex rules, with money supply and government spending as controls and one or more state variables as target variables.

Finally, you may also define a rule in which the real interest rate - instead of the money supply (as was the case in the experiment presented in this chapter) - is used to respond to changes in prices. This type of rule is typically used by many researchers - e.g. Taylor (1998) and Woodford (2003) to discuss monetary policy rules in the U.S. To do so, notice that you will have to redefine the interest rate as a control variable and the money supply as a state variable, since when the interest rate is used as a control, the money supply becomes an endogenous variable. Since this is a substantial change in the model structure, it may require you to start from scratch to input the new model in Duali.

7. Further Reading

The prototype Taylor model presented in this chapter, together with U.S. and multicountry extended econometric versions of it are developed in Taylor (1993). Holly and Hughes-Hallett (1989, Chapter 7) provide an introduction to the application of optimal control techniques to rational expectations models. Amman and Kendrick (1996), (1999c), (2000), (2003) develop optimal control techniques and applications for a variety of rational expectations models. Taylor (1998) and Woodford (2003) provide a wide treatment of the application of policy rules to rational expectations models. For a useful starting point to coming abreast of recent work on the recent variety of optimizing trend-deviation macroeconomic models see Kozicki and Tinsley (2002).

For discussion of the robust control approach to stochastic control see Deissenberg (1987), Rustem (1992), Hansen and Sargent (2001) and Rustem and Howe (2002)

Appendix A

Running GAMS

This appendix provides the details for running the GAMS software on a PC. In order to use GAMS with other input files substitute the appropriate file name for `trnsport.gms` in the following. For help and information about obtaining GAMS go the GAMS Development Corporation web site at

http://www.gams.com

There is a student version of GAMS that can be downloaded and that can solve all or almost all of the models used in this book. It the model is too large, usually a small change in the number of time periods or some other set is sufficient to reduce the size so that it will run on the student version.

- Go to the book web site at

  http://www.eco.utexas.edu/compeco

  and to the "Input Files for Chapters in the Book" section of the web site. Right click on the `trnsport.gms` filename and select the "Save Target As …" option in order to save the file in your preferred directory.

- Chose Programs from the Start menu and then chose GAMS and gamside. Chose Open from the File menu, navigate to the `trnsport.gms` file and open it for editing. Notice in the complete GAMS statement of the model that, as is the usual case in GAMS, the model is defined in steps

  first the sets
  then the parameters
  then the variables
  then the equations
  and finally the model and solve statements.

- Solve the model by choosing Run from the File menu and then check the solution log to be sure that you have

  SOLVER STATUS: 1 NORMAL COMPLETION

  and

  MODEL STATUS: 1 OPTIMAL

  Then close the log file window.
- Click on the `trnsport.lst` file window and scroll through this listing file to see the solution. Note that the `*.lst` file extension used here is an abbreviation for a "listing" of the output file.

Notice that the GAMS output has the following structure:

Echo Print: shows a listing of the input file with the line numbers added.

Error Messages: in the case of errors in the input file, they will be signaled by GAMS with "****" on the leftmost part of the corresponding line of input where the error was found, and with "$number" just below the part of the line of input where the error is located, where "number" will contain a specific error code. Then, at the end of the list of the input file, GAMS will display the explanation of each of the error codes found.

Equation Listing: shows each equation of the model with the corresponding values for sets, scalars and parameters.

Column Listing: shows a list of the equations' individual coefficients classified by columns.

Model Statistics: shows information such as model number of equations, number of variables, etc.

Solve Summary: shows information such as solver and model status at the end of the GAMS run, etc.

Solution Listing: shows the solution values for each equation and variable in the model. Each solution value is listed with four pieces of information, where a dot "." means a

value of zero, EPS a value near zero, and +INF and –INF mean plus or minus infinite respectively:

LOWER (the lower bound)
LEVEL (the solution level value)
UPPER (the upper bound)
MARGINAL (the solution marginal value; it corresponds, for linear or nonlinear programming problems, to the increase in the objective value due to a unit increase in the corresponding constraint)

Report Summary: shows the count of rows or columns that are infeasible, non-optimal or unbounded.

Appendix B

Running Mathematica

Mathematica is a widely available commercial software system. A web site for information about it is

http://www.wolfram.com

- Choose Programs from the Start menu and then choose Mathematica. Wait for a few seconds, and a new window with a menu bar in its upper part will appear. The content of this window will be a white sheet called `Notebook`, which is like a document in a standard word processor. If you specified a file when opening Mathematica, this file will be displayed.

- Select Getting Started from the Help menu (located in the upper right corner of the Mathematica window) and read the information that will appear in the "Info dialog box". To begin practicing with Mathematica, perform the calculations suggested in the section "Doing Calculations". While doing this, you will appreciate the basic way of working with Mathematica, i.e. your Notebook will successively display your inputs and the corresponding outputs. You may also notice that on the right side of your Notebook, a hierarchy of brackets appears. Each of them defines a cell (or a group of cells) which are the basic units of organization in a Notebook. As you will quickly realize, cells can be hierarchically arranged (as in set and subsets). There are different kinds of cells: they can contain text, Mathematica input, Mathematica output, or graphs, etc. Different small characters within each bracket identify the kind of cell. Here are some basic things you can do with cells:

  - To edit a cell (that is, to be able to work with it) just click on its bracket.
  - To edit a group of cells, just click and drag on their brackets.
  - To find out or change the kind of cell, edit the cell, select "Style" in the main menu and choose your option.

- To divide or merge cells, take the cursor to the division/insertion point of your choice, select "Cell" in the main menu and choose your options.
- To run a portion of a program contained in a cell of group of cells, just edit the cells and select Action-Evaluate in the main menu (or just press **"**Shift-Enter**"**)
- Finally, to save your Notebook, select "File" in the main menu and choose your options.
- Go to the book web site at http://www.eco.utexas.edu/compeco and then to the "Input Files for Chapters in the Book" section of the web site. Right click on the `Leontief.nb` filename and select the "Save Target As …" option in order to save the file in your preferred directory.
- Go to "File" menu and click Open to open the file**.**
- To run an input command or a cell containing a series of commands, click on the bracket on the right of it and hit Shift+Return (hold the shift key and hit Return at the same time). The output will be displayed following the input, *unless* there is a "**;"** at the end of the input command line ("**;"** suppresses the output). You can run multiple cells by highlighting the corresponding brackets with your mouse and hitting Shirt+Return once.
- Modify commands and re-run them sequentially, cell after cell, so that you can see the changes in the corresponding outputs.
- If you either select the outer most bracket and press Shift-Enter, or go to the Kernel menu, choose Evaluation**,** and Evaluate Notebook**,** you will re-run the complete program. If your program is large this may take a few minutes and it may be difficult for you to track down the results of your modifications. On the other hand, sometimes your modifications may require an updating of previous results, a clearing of previous values or a change of attributes (and the Clear command or the SetAttributes command are usually at the beginning of the program). In these cases you may need to re-run the complete program to avoid errors or spurious results.

Appendix C

Running the Solver in Excel

Download the file for the growth model or for the neural net from the book web site. Your version of Excel may not have the Solver option available by default. To check this look for the Solver option on the Tools menu. If you don't find it, click Add-Ins**,** check Solver Add-in, and click OK. Then look in the Tools menu again.

Appendix D

Ordered Sets in GAMS

As was discussed above in the chapter on Macroeconomics in GAMS, the definition of lagged indices for variables in GAMS may be somewhat problematic if it is not done with care. For example, if the variables "w" and "z" were defined over a set "t" (i.e. w(t) and z(t)) such as

$$t = \{0,1,2,3\}$$

then an expression like

$$eq(t)..\ w(t) =E= z(t-1)$$

will result in the following equations being generated by GAMS

$$eq(0)..\ w(0) =E= 0;$$
$$eq(1)..\ w(1) =E= z(0);$$
$$eq(2)..\ w(2) =E= z(1);$$
$$eq(3)..\ w(3) =E= z(2);$$

Thus, it will cause GAMS to assign the value zero to the first element of w(t), since the element "z(-1)" of the variable "z" is not defined. We do not want this to happen, since it will be a source of confusion at the time of assigning initial values for lagged variables and also for the interpretation of solution values corresponding to the initial periods of the solution horizon.

To be sure about the results of the dynamic specifications in GAMS, every time one writes a program involving dynamic variables it is advisable to set OPTION LIMROW equal to the maximum number of periods involved in the solution of the model. This will tell GAMS to print a detailed equation-by-equation solution report which will allow one to check period-by-period the evolution of the time indices for each variable within each equation. It is particularly important to check the specification of the equation for the first few and the last few time periods. For example, here is how the corresponding GAMS output looks for equation "eq6" in the chapter on Macroeconomics in GAMS.

```
eq6(t+2)..   piex(t+2) =E=    alpha * pi(t+1) + beta * pi(t) ;
```

when Hall and Taylor's model is solved for a time horizon of 7 periods - that is, for a "t" set equal to {0,1, … , 5,6}.

```
---- EQ6        =E=  expected inflation

EQ6(2)..  - 0.2*PI(0) - 0.4*PI(1) + PIEX(2) =E= 0 ;

EQ6(3)..  - 0.2*PI(1) - 0.4*PI(2) + PIEX(3) =E= 0 ;

EQ6(4)..  - 0.2*PI(2) - 0.4*PI(3) + PIEX(4) =E= 0 ;

EQ6(5)..  - 0.2*PI(3) - 0.4*PI(4) + PIEX(5) =E= 0 ;

EQ6(6)..  - 0.2*PI(4) - 0.4*PI(5) + PIEX(6) =E= 0 ;
```

Notice that eq6(t+2) goes from periods 2 to 6, while pi(t) goes from 0 to 4, pi(t+1) from 1 to 5 and piex(t+2) from 2 to 6. This means that the effective solution horizon for the model was equal to 5 periods, 2 less than the number of elements of the set "t".

For further details see the chapter on "Set as Sequences: Ordered Sets" in the GAMS User's Guide at www.gams.com

Appendix E

Linearization and State-Space Representation of Hall and Taylor's Model

The linearization method that we will use is known as Johansen's method – see Johansen (1960). It involves transforming all the variables in the model into percentage changes with respect to a base case. We introduced this method in the chapter on General Equilibrium Models. There we learned that there are some rules, analogous to differentiation, which simplify the task of linearizing a model. We will apply those rules here.

Remember that since the Hall and Taylor model is a dynamic model, all its variables have an explicit or implicit time subscript. It is important to understand that the percentage changes of each variable will be changes with respect to a baseline case (the point of linearization) and not with respect to "the previous period". If our baseline case is the steady-state and, say, X^*_{t+4} takes the value 0.01, this means that the variable *X*, at time *t+4*, is 1% higher than its steady-state value. It does not mean that X^*_{t+4} is 1% higher than X^*_{t+3}.

The steady-state solution for Hall and Taylor's original nonlinear model in levels is: $Y = 6000$, $R = 0.05$, $plev = 1$ and $E = 1$. These steady-state values correspond to the following values for policy and exogenous variables: $M = 900$, $G = 1200$, $YN = 6000$ and $plevw = 1$. We will pick the steady-state solution as our baseline or point of linearization. Thus, the expressions in the sum rule in the General Equilibrium Models chapter for

$$X = Y + Z$$

becomes

$$X^* = s_y Y^* + s_z Z^*$$

where X^*, Y^* and Z^* are percentage deviations of the corresponding level variables and s_y and s_z are the shares

$$s_y = \frac{Y_{ss}}{Y_{ss} + Z_{ss}} \text{ and } s_z = \frac{Z_{ss}}{Y_{ss} + Z_{ss}}.$$

where the subscript "*ss*" means "steady-state value".

The original twelve-equation model contains the equations listed below:

IS-LM

(1) GDP identity $Y = C + I + G + X$

(2) Disposable Income $Y^d = (1-t)Y$

(3) Consumption $C = a + bY^d$

(4) Investment $I = e - dR$

(5) Money Demand $M/P = kY - hR$

Expectations Augmented Phillips Curve

(6) Expected Inflation $\pi^e = \alpha\pi_{-1} + \beta\pi_{-2}$

(7) Inflation Rate $\pi = \pi^e + f\{(Y_{-1} - Y_N)/Y_N\}$

(8) Price Level $P = P_{-1}(1+\pi)$

Foreign Sector

(9) Real Exchange Rate $EP/P_W = q + vR$

(10) Net Exports $X = g - mY - nEP/P_W$

Government Deficit and Unemployment

(11) Government Deficit $G_d = G - tY$

(12) Unemployment Rate $U = U_N - \mu\{(Y - Y_N)/Y_N\}$

To obtain the equation for Y^* (that is, GDP percent deviation from steady-state), we substitute eqs. (2), (3), (4) and (10) into eq. (1). Linearizing, we obtain

(e.1) $Y^* = -sa_{12}R^* - sa_{13}plev^* - sa_{14}E^* + sb_{12}G^* + sc_{12}plevw^*$

where[61]

$aux = (1 - (b\ (1 - t) - n))$

$sa_{12} = (d\ R_{ss})/(aux\ Y_{ss})$

$sa_{13} = (n\ E_{ss}\ plevw_{ss}\ plev_{ss})/(aux\ plevw_{ss}^2\ Y_{ss})$

$sa_{14} = (n\ plev_{ss}\ plevw_{ss}\ E_{ss})/(aux\ plevw_{ss}^2\ Y_{ss})$

$sb_{12} = G_{ss}/(aux\ Y_{ss})$

$sc_{12} = (n\ E_{ss}\ plev_{ss}\ plevw_{ss})/(aux\ plevw_{ss}^2\ Y_{ss})$

[61] The reason why we define the coefficients as sa_{12}, etc., will become clear below, when we write the model in matrix notation.

To derive the equation for R^* (Real Interest Rate), linearizing and re-arranging eq. (5) we obtain

(e.2) $$R^* = -sa_{21}\ Y^* - sa_{23}\ plev^* + sb_{21}\ M^*$$

where

$$sa_{21} = -(k\ Y_{ss})/(h\ R_{ss});\quad sa_{23} = -M_{ss}\ plev_{ss}/(h\ plev_{ss}^{\ 2}\ R_{ss});\quad sb_{21} = -M_{ss}/(h\ plev_{ss}\ R_{ss}).$$

To obtain the equation for $plev^*$ (Domestic Price Level), substitute equation (6)

(6) $$\pi^e = \alpha\pi_{-1} + \beta\pi_{-2}$$

into equation (7)

(7) $$\pi = \pi^e + f\left\{\left(Y_{-1} - Y_N\right)/Y_N\right\}$$

to get

(e.3) $$\pi = \alpha\pi_{-1} + \beta\pi_{-2} + f(Y_{-1} - YN)/YN$$

This expression combines variables in levels and variables in rates of growth. To avoid the confusion that may arise from working with percentage changes of rates of growth, we proceed as follows. Taking into account that the percent deviation of a variable is, for small deviations, approximately equal to its corresponding log difference, we can write

(e.4) $$(Y_{-1} - YN)/YN \approx \ln Y_{-1} - \ln YN$$

Now, we can re-write eq. (8), i.e.

$$P = P_{-1}(1+\pi)$$
$$1+\pi = \frac{P}{P_{-1}}$$
$$\pi = \frac{P - P_{-1}}{P}$$

as

(e.5) $$\pi = (plev - plev_{-1})/plev_{-1}$$

and applying the same property as above, we can write

(e.6) $$\pi \approx \ln plev - \ln plev_{-1}$$

and then

(e.7) $$\pi_{-1} \approx \ln plev_{-1} - \ln plev_{-2}$$

(e.8) $$\pi_{-2} \approx \ln plev_{-2} - \ln plev_{-3}$$

Now, substituting (e.4) and (e.6)-(e.8) into (e.3) and linearizing we obtain

(e.9) $$plev^* = sa1_{31} Y^*_{-1} + sa1_{33}\ plev^*_{-1} + sa2_{33}\ plev^*_{-2} + sa3_{33}\ plev^*_{-3} + sc_{31}\ YN^*$$

where

$$sa1_{31} = f\ ;\ sa1_{33} = 1+\alpha\ ;\ sa2_{33} = \beta - \alpha\ ;\ sa3_{33} = -\beta\ ;\ sc_{31} = -f.$$

Finally, to derive the equation for E^* (Nominal Exchange Rate), linearizing eq. (9) we obtain

(e.10) $$E^* = -sa_{42}\ R^* - sa_{43}\ plev^* + sc_{42}\ plevw^*$$

where

$$sa_{42} = -v\ \ plevw_{ss}\ \ R_{ss}/(plev_{ss}\ \ E_{ss});\ \ sa_{43} = 1;\ \ sc_{42} = 1.$$

Since variables G_d (Government deficit) and U (unemployment rate) do not have any "feedback" with the other equations in the model, we can ignore eqs. (11) and (12). In summary, the four equations of our model are (e.1), (e.2), (e.9) and (e.10), i.e.

(e.1) $Y^* = -sa_{12}R^* - sa_{13}plev^* - sa_{14}E^* + sb_{12}G^* + sc_{12}plevw^*$

(e.2) $R^* = -sa_{21}\ Y^* - sa_{23}\ plev^* + sb_{21}\ M^*$

(e.9) $plev^* = sa1_{31}Y^*_{-1} + sa1_{33}\ plev^*_{-1} + sa2_{33}\ plev^*_{-2} + sa3_{33}\ plev^*_{-3} + sc_{31}\ YN^*$

(e.10) $E^* = -sa_{42}\ R^* - sa_{43}\ plev^* + sc_{42}\ plevw^*$

Notice that since in this linearized representation all variables are in percent deviations, their steady-state values will all be zeroes.

Writing our structural model in matrix notation, we obtain

(e.11) $$SA\ X = SA1\ X_{-1} + SA2\ X_{-2} + SA3\ X_{-3} + SB\ U + SC\ V$$

where:

$$X = \begin{bmatrix} Y^* \\ R^* \\ plev^* \\ E^* \end{bmatrix} \qquad U = \begin{bmatrix} M^* \\ G^* \end{bmatrix} \qquad V = \begin{bmatrix} YN^* \\ plevw^* \end{bmatrix}$$

and

$$SA = \begin{bmatrix} 1 & sa_{12} & sa_{13} & sa_{14} \\ sa_{21} & 1 & sa_{23} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & sa_{42} & sa_{43} & 1 \end{bmatrix} \qquad SA1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ sa1_{31} & 0 & sa1_{33} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$SA2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & sa2_{33} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad SA2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & sa3_{33} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$SB = \begin{bmatrix} 0 & sb_{12} \\ sb_{21} & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad SC = \begin{bmatrix} 0 & sc_{21} \\ 0 & 0 \\ sc_{31} & 0 \\ 0 & sc_{42} \end{bmatrix}$$

We have obtained above a structural model which is, of course, a simultaneous system of equations. To obtain its reduced form, we have to get rid of this simultaneity and to express each endogenous variable as only a function of policy, exogenous and pre-determined variables. This can be done easily.

From equation (e.11) the reduce form can be obtained as

(e.12) $X = RA1\, X_{-1} + RA2\, X_{-2} + RA3\, X_{-3} + RB\, U + RC\, V$

where:

$RA1 = SA^{-1}\, SA1;\ RA2 = SA^{-1}\, SA2;\ RA3 = SA^{-1}\, SA3;\ RB = SA^{-1}\, SB;\ RC = SA^{-1}\, SC$

Equation (e.12) is a third-order system difference equation (the maximum lag is equal to 3). It is necessary to reduce it to a first order system that is called the "state-space" representation.[62] For instance, to analyze some dynamic properties of the linearized model, we have to know its characteristic roots, and these are equal to the eigenvalues of the matrix of the first order version (matrix *A* below).[63] Also, to determine the model controllability or to perform policy experiments with Duali, the input model has to be in state-space form. To make this transformation, we augment the state variable by taking the following steps. We define the new vectors XL_{-1} and XLL_{-1} as

[62] The concept of state-space goes beyond this, but we will not deal with it here.

[63] See Mercado and Kendrick (1999).

(e.13) $$XL_{-1} = \begin{bmatrix} xlY^*_{-1} \\ xlR^*_{-1} \\ xlplev^*_{-1} \\ xlE^*_{-1} \end{bmatrix} = X_{-2} = \begin{bmatrix} Y^*_{-2} \\ R^*_{-2} \\ plev^*_{-2} \\ E^*_{-2} \end{bmatrix}$$

(e.14) $$XLL_{-1} = \begin{bmatrix} xllY^*_{-1} \\ xllR^*_{-1} \\ xllplev^*_{-1} \\ xllE^*_{-1} \end{bmatrix} = XL_{-2} = \begin{bmatrix} xlY^*_{-2} \\ xlR^*_{-2} \\ xlplev^*_{-2} \\ xlE^*_{-2} \end{bmatrix} = X_{-3} = \begin{bmatrix} Y^*_{-3} \\ R^*_{-3} \\ plev^*_{-3} \\ E^*_{-3} \end{bmatrix}$$

Then, re-write (e.12) as

(e.15) $$X \;=\; RA1\, X_{-1} \;+\; RA2\, XL_{-1} \;+\; RA3\, XLL_{-1} \;+\; RB\, U \;+\; RC\, V$$

Define the augmented state vector x

(e.16) $$x = \begin{bmatrix} X \\ XL \\ XLL \end{bmatrix},$$

re-write (e.13) and (e.14) as

(e.15) $$XL \;=\; X_{-1}$$

(e.16) $$XLL \;=\; XL_{-1} \;=\; X_{-2}$$

and finally transform (e.15) into its state-space representation as

(e.17) $$x \;=\; A\, x_{-1} \;+\; B\, U \;+\; C\, V$$

where U and V are the same as above, where

$$A = \begin{bmatrix} RA1 & RA2 & RA3 \\ I & 0 & 0 \\ 0 & I & 0 \end{bmatrix} \quad B = \begin{bmatrix} RB \\ 0 \\ 0 \end{bmatrix} \quad C = \begin{bmatrix} RC \\ 0 \\ 0 \end{bmatrix}$$

and where *I* is a (4x4) identity matrix and 0 are (4x4) and (4x2) matrices of zeros as appropriate.

In Hall and Taylor's model, the policy variables contemporaneously affect the model's endogenous variables, and this is also true for its "state-space" representation. In order to obtain a proper state-state representation, that is, one in which the control variables also appear with one lag, we have to assume that there is a one-period delay between a policy decision and its implementation. Then, we can substitute M_{-1}^{*} for M^{*}, and G_{-1}^{*} for G^{*}. We will also assume that the exogenous variables YN^{*} and $plevw^{*}$ affect the system with one lag instead of contemporaneously. Expressing the model in this way, we can make use of many results from the optimal control literature, which works with models with one-lag controls. Also, the Duali software works in this way.

Thus, in matrix notation, with numerical parameter values derived from the corresponding original model parameter values, and where all the variables are percent deviations from the steady-state, the state-space representation of Hall and Taylor's model can then be written as in equation (1) in the chapter on Stochastic Control in Duali.

Appendix F

Introduction to Nonlinear Optimization Solvers

Solving nonlinear optimization problems usually requires the use of numerical methods. In general, those methods consist of a "smart" trial and error algorithm that is a finite sequence of computational steps designed to look for convergence to a solution. There is a variety of algorithms to solve nonlinear problems. Some of them are global methods, in the sense that they perform a parallel exploration of many regions of the optimization space. One example of this type of solution method is genetic algorithms. Other methods are local methods, since they tend to focus on the exploration of a particular region of the optimization space. In this appendix we will introduce two of the most popular local methods: the gradient method and the Newton method. Varieties of these methods are used by the solvers in Excel, GAMS and MATLAB. Before introducing the gradient method and the Newton method, we begin with a simple example.

Suppose that we are trying to find the maximum of a nonlinear function

(1) $$y = f(x)$$

such as the one represented in Figure F.1 below.

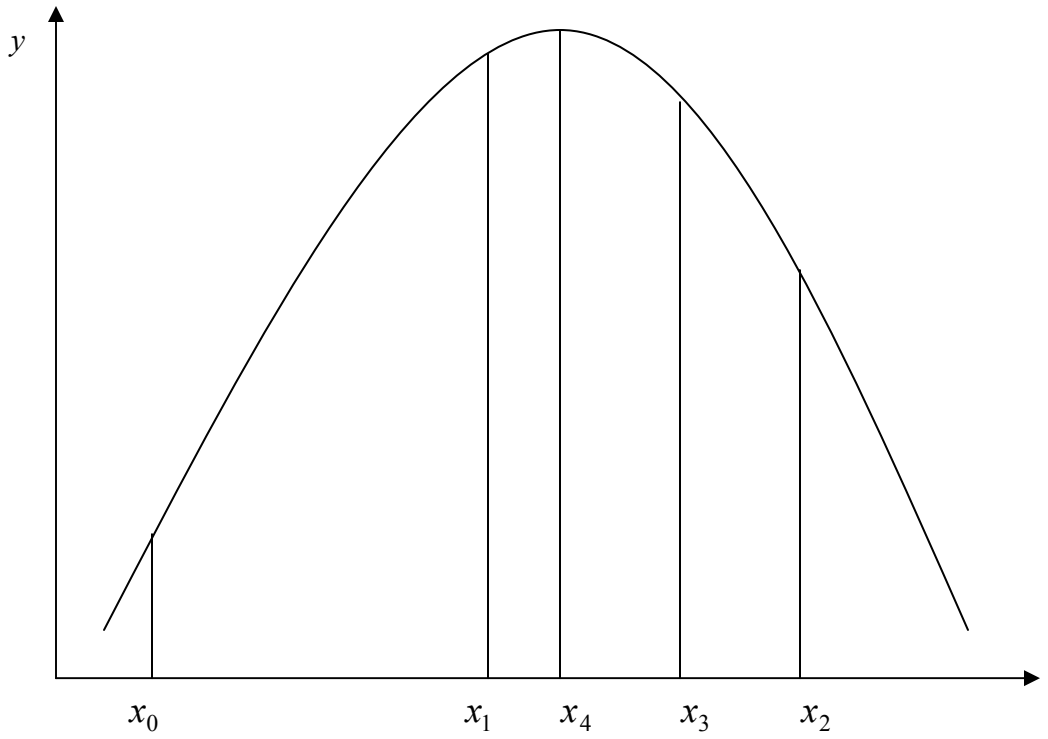


Figure F.1 A Nonlinear Function

A simple and very rudimentary algorithm to find the solution could be as follows. We choose an arbitrary initial value for *x*, such as x_0 in Figure F.1, and compute the corresponding $y_0 = f(x_0)$. Then we increase that value by a constant magnitude *h* (we name this magnitude the "search step") that we also choose in an arbitrary way. For the new value of *x*, that is x_1, we compute the corresponding value of

(2) $$y_1 = f(x_1) = f(x_0 + h)$$

and we compare this value to the one obtained in the previous step. We continue to do this as long as the differences between two successive values of *y* are positive (negative for a minimization problem). As soon as we compute a difference with a negative sign (in Figure F.1 this would correspond to x_2), we reverse the direction of the search. We begin to move in the opposite direction along *x* (that is, subtracting *h* from *x*) and we use for *h* a smaller value than the one we were using while we moved in the opposite direction. We continue like this until we find again a difference between two successive

values of *y* which is negative. We then again reverse the direction of the search and we reduce once more the size of *h*. And so on. We stop when the difference between two successive values of *y* falls below a pre-established tolerance limit.

The gradient method and the Newton method are iterative methods like the one presented above. However, they exploit local information about the form of the function. That is, they use the function's derivatives. To illustrate this we change to a multivariate example. In this case we use the following equation to obtain each new value of the vector *x*

(3) $$x_{n+1} = x_n + h\Delta x$$

where *h* is the search step - now always a positive value - and where Δx is the direction of change which, as we will see, will be determined by the function's derivatives.

The gradient method uses the first derivatives or gradient, which give us information about how the function changes in the neighborhood of a given point. Its basic framework is the well known first order Taylor approximation

(4) $$f(x_{n+1}) \cong f(x_n) + h\nabla f(x_n)\Delta x$$

where $\nabla f(x_n)$ is the gradient vector. Notice that since *h* is supposed to be positive, the best direction of motion will be

(5) $$\Delta x = \nabla f(x_n)$$

for a maximization problem, since

(6) $$f(x_{n+1}) \cong f(x_n) + h\nabla(f(x_n))^2 > f(x_n)$$

Also, for a minimization problem

(7) $$\Delta x = -\nabla f(x_0)$$

since

(8) $$f(x_{n+1}) \cong f(x_n) - h\nabla(f(x_n))^2 < f(x_n)\,.$$

The basic framework of the Newton method is the second order Taylor approximation

$$f(x_{n+1}) \cong f(x_n) + h\nabla f(x_n)\Delta x + \frac{h}{2}\Delta x' H(x_n)\Delta x \tag{9}$$

where $H(x_0)$ is the second order derivative or Hessian which tells us how the slope of the function changes in a neighborhood of a given point.

Assuming the Taylor expansion of a function f is a good global approximation to that function, we will approximate the optimum value of f by optimizing its Taylor expansion. In our case, this is equivalent to saying that to determine the best direction of motion Δx we have to optimize the expression (9). Differentiating (9) with respect to Δx, making the result equal to zero and solving for Δx we obtain

$$\Delta x = -\frac{\nabla f(x_n)}{H(x_n)} \tag{10}$$

which will be the best direction of motion for Newton's method.

Sometimes iterative methods like the ones presented above do not converge to a solution after a finite number of iterations. This problem may be overcome by changing the maximum number of iterations, or the size of the search step, or the tolerance limit or the initial value of the search. Most solvers allow you to change these parameters.

Notice also, as is the general case for numerical methods dealing with nonlinear optimization problems, that if there is more than one local optimum we will find only one of them. Thus, we will never know for sure if the optimum we reached was a local or a global one. A rough way of dealing with this problem is to solve the problem providing the algorithm with alternative initial values of the search.

In this appendix we presented three numerical methods of increasing complexity. Of course, the more complex ones make use of more information thus reducing, in general, the number of steps to achieve convergence. However, those steps become more complex, since they required the computation of a gradient or a Hessian. Then, there are trade offs to be evaluated when choosing a solution method.

There are additional methods to solve nonlinear problems numerically - i.e. conjugate gradient method, penalty function method, sequential quadratic programming, etc. - a number of which extend, combine or mimic the ones introduced here. For a

comprehensive presentation you are referred to Judd (1998) and Miranda and Fackler (2002). The Excel Solver uses a conjugate gradient method or a Newton method. GAMS uses a variety of methods, depending on the solver you choose or have set up as the default nonlinear solver. The MATLAB solver used in the Portfolio Model in MATLAB chapter and invoked by the `fmincon` function uses a sequential quadratic programming method. For details on the specific methods used by Excel, GAMS and MATLAB you are referred to their corresponding user's and solver's manuals.

Appendix G

Linear Programming Solvers

A linear programming problem is one of maximizing a linear objective function subject to a set of linear constraints. In economics, it is also frequently required that the variables of the problems be nonnegative. Thus, in mathematical terms a linear programming problem can be expressed as

$$\begin{aligned} &\max y = b'x \\ &s.t. \quad Ax \le k \\ &\qquad x \ge 0 \end{aligned}$$

where y is a scalar, x is a vector of variables, b and k are vectors of constants and A is a matrix. If the problem is one of minimization, it can be written as one of maximizing the objective function with a negative sign, and changing the direction of the inequalities by multiplying both sides by minus one. To have an intuitive graphical representation of the problem, suppose that we have a problem with two variables and three restrictions, i.e.

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \qquad A = \begin{bmatrix} a_{11}\ a_{12} \\ a_{21}\ a_{22} \\ a_{31}\ a_{32} \end{bmatrix} \qquad \text{and} \qquad k = \begin{bmatrix} k_1 \\ k_2 \\ k_3 \end{bmatrix}.$$

Thus, the problem can be represented as in Figure G.1 below.

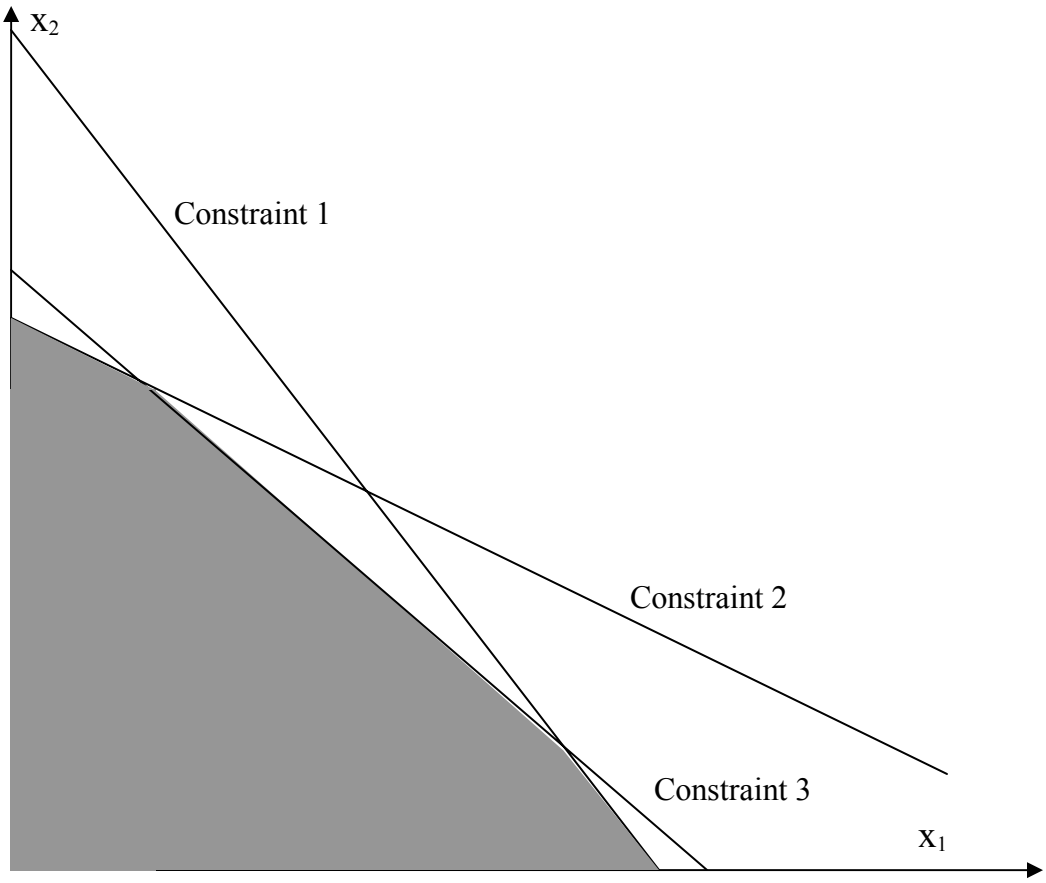


Figure G.1 Feasible Solutions Set

We can see that the problem constraints define an area - the shaded one - that contains all the feasible solutions. It is a closed, convex and lower-bounded set, also known as a simplex. In Figure G.2 below we added the corresponding level curves of the objective function. Since for this example

$$y = b_1 x_1 + b_2 x_2$$

then those level curves are given by

$$x_2 = \frac{y}{b_2} - \frac{b_1}{b_2} x_1$$

We will have one level curve for each value of y.

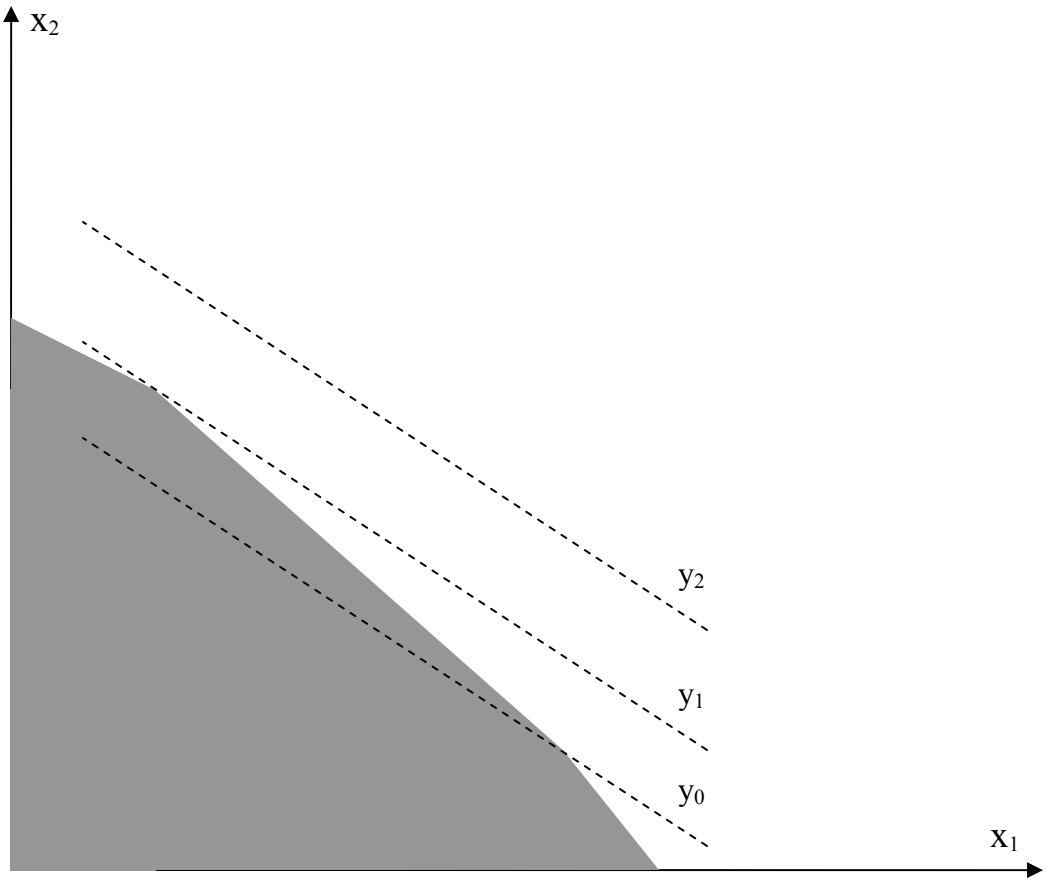


Figure G.2 Level Curves

We can see that the maximum feasible *y* will be y_1. Generalizing, we can say that the optimum value of a linear programming problem will be obtained at the point in which a level curve is tangent to the simplex of feasible solutions. And this will always happen at a vertex of the simplex. Notice also that multiple - actually an infinite number - of solutions will be obtained when the level curve is tangent to a segment between two vertices.

Thus, a solution method could be one that focuses on the evaluation of the vertices of the simplex of feasible solutions. A rudimentary method would evaluate all vertices and choose the one that generates the highest value - for a maximization problem - of the objective function. However, the number of vertices grows very quickly as the number of variables and constraints increases.

A more efficient method, used by the default GAMS solver BDMLP, is the iterative procedure known as the simplex method. Starting from a given vertex, this

method looks for the best direction of motion toward another vertex. To do so, it starts by transforming the inequality restrictions into equalities by means of the addition of new nonnegative variables known as "slack variables". In our two-variable three-restriction example, this is equivalent to writing the new constraints as

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + x_3 &= k_1 \\ a_{21}x_1 + a_{22}x_2 + x_4 &= k_2 \\ a_{31}x_1 + a_{32}x_2 + x_5 &= k_3 \end{aligned}$$

or in matrix notation

$$[A\ I]x = k$$

where I is a 3x3 identity matrix and where the vector x is now

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}.$$

Notice that the new matrix $[A\ I]$ is a 3x5 matrix. Thus, if we set to zero any two variables in x we will be left with a 3x3 matrix and a 3x3 system of linear equations. This system will have a solution if the corresponding row (columns) vectors in the matrix are linearly independent. Also that solution, which corresponds to the base of the tri-dimensional space spanned by those vectors, will be a vertex of the simplex of feasible solutions. Thus, we will name that solution the "basic feasible solution".

The next step in the simplex method is to evaluate the solution to check if we are at the optimum. To do so, we compute

$$\frac{\partial y}{\partial x_{NB}}$$

where x_{NB} are the non-basic variables. If any one of these derivatives is greater than zero, we are not at the optimum since the objective function could be incremented by increasing the corresponding non-basic variable. The next step is thus to move to another

vertex incorporating this variable into the base and deleting one of the variables previously in the base. The selection of the basic variable to be deleted is more involved. Ideally, we should delete the variable which constraints the most the potential increase in the objective expected from incorporation of the new basic variable. To do so, the constraints have to be re-written now with the basic variables as functions of the non-basic ones, and the resulting system has to be analyzed.

We then continue evaluating the objective function and incorporating-deleting variables to the basic solution until we reach an optimum.

For more detailed presentations of the simplex method, you are referred to Chiang (1984), Rardin (1998) and Silverberg and Suen (2001). For details on GAMS linear programming solvers, see the corresponding GAMS Solvers manuals at http://www.gams.com.

Appendix H

The Stacking Method in GAMS

As a compact way of writing a multi-equation model, GAMS allows us to write indexed equations. As seen in a number of chapters of this book those indexes may represent commodities, locations, time periods, etc.

For example, the equations corresponding to a problem such as

$$\max J = \sum_{i=0}^{2} w_1 x_i + w_2 y_i$$

$$s.t. \quad (1) \quad a_{11} x_i + a_{12} y_i = b_1$$

$$(2) \quad a_{21} x_i + a_{22} y_i = b_2$$

can be represented in GAMS as

```
eqj..       j =e= sum(i, w1 * x(i) + w2 * y(i));
eq1(i)..    a11 * x(i) + a12 * y(i)) =e= b1;
eq2(i)..    a21 * x(i) + a22 * y(i)) =e=  b2;
```

When the index set is $i = \{0,1,2\}$ the model will be expanded and stacked in the following way

```
j =e= w1*x(0) + w2*y(0) + w1*x(1) + w2*y(1) + w1*x(2) + w2*y(2)
eq1(0)..    a11 * x(0) + a12 * y(0)) =e= b1;
eq2(0)..    a21 * x(0) + a22 * y(0)) =e= b2;
eq1(1)..    a11 * x(1) + a12 * y(1)) =e= b1;
eq2(1)..    a21 * x(1) + a22 * y(1)) =e= b2;
eq1(2)..    a11 * x(2) + a12 * y(2)) =e= b1;
eq2(2)..    a21 * x(2) + a22 * y(2)) =e= b2;
```

Notice that before we had a model with an objective function and two indexed equations and two variables (`x(i)` and `y(i)`) and now we have a model with one objective function, six equations and six variables (`x(0), x(1), x(2), y(0), y(1) and y(2)`). Thus, before solving the model, GAMS transforms a model of *n* indexed

equations into one of *n x card* equations plus the objective function, where *card* indicates the number of elements in the index set. If the index denotes time periods, this is equivalent to transforming a dynamic model with *n* indexed equations and *t* time periods into an equivalent static model of *n x t* equations plus the objective function.

When, as in the chapters on General Equilibrium Models in GAMS and Macroeconomics in GAMS, we are interested in solving a system of equations and not an optimization problem, we just set the objective function equal to any constant value (i.e. `j =e= 0;`). Thus, when executing the corresponding solver statement, i.e.

```
solve model maximizing j using nlp;
```

GAMS will expand and stack the system of equations and it will solve it as a by product of a "pseudo-optimization".

Appendix I

Running MATLAB

This appendix provides the details for running the MATLAB software on a PC to solve the portfolio model. In order to use MATLAB with other input files substitute the appropriate file name for `mcportfol.m` in the following.

For help and information about obtaining MATLAB go to The MathWorks web site at

http://www.mathworks.com

- Go to the book web site at

  http://www.eco.utexas.edu/compeco

  and to the" Input Files for Chapters in the Book" section of the web site. *Right* click on the mcportfol.m filename and select the "Save Target As …" option in order to save the file in your preferred directory.
- Chose Programs from the Start menu and then chose MATLAB.
- In the Current Directory section of the main MATLAB window click on the icon that contains "…" in order to browse to the folder where you stored the `mcportfol.m` file. Then double click on the `mcportfol.m` filename.
- A window that contains the `mcportfol.m` file will open. In order to solve the model pull down the Debug menu and select the Run option. A graph will appear showing the results of the runs.
- In order to see the numerical results select the MATLAB main window and look in the Command Window section.
- If you run a MATLAB program that uses a number of functions stored in separate files (such as the portfolio.m or the models in the genetic algorithm chapters or in the agent-based model chapter) make sure you download all those files in the same directory.

Appendix J

Obtaining the Steady-State of the Growth Model

Here we are interested in deriving the steady-steady solution of the model presented in the Growth in Excel chapter. More detailed derivation steps can be found in Azariadis(1993), Sections 7.3 and 13.4.

We begin by defining the utility function

(1) $$u(C_t) = \frac{1}{1-\tau} C_t^{1-\tau}$$

and the production function

(2) $$f(K_t) = \theta K_t^{\alpha}.$$

Thus, as in the Growth in Excel chapter, the model we want to solve can be stated as find

$(C_0, C_1, \cdots, C_{N-1})$ to maximize

(3) $$J = \sum_{t=0}^{\infty} \beta^t u(C_t)$$

subject to

(4) $$K_{t+1} = K_t + f(K_t) - C_t$$

(5) K_0 given.

(6) $$\lim_{t \to \infty} \left[\beta^t u'(C_t) K_t \right] = 0$$

where (3) is the criterion function, (4) is the capital accumulation equation and (5) is the initial condition. Since we are now interested in deriving the steady-state solution of the model, we consider an infinite horizon problem. Thus, instead of a fixed terminal condition, we now impose the transversality condition (6), where $u'(C_t)$ is the derivative of the utility function. This condition states that the discounted lifetime utility is

maximal when the capital stock is zero or, in other terms, that at time *t* the present value of capital tends to zero as time goes to infinity.

Re-arranging (4) and substituting for C_t in (3) we obtain

$$J = \sum_{t=0}^{\infty} \beta^t v(K_t, K_{t+1}) \tag{7}$$

where

$$v(K_t, K_{t+1}) = u\left[f(K_t) + K_t - K_{t+1}\right]. \tag{8}$$

Differentiating (7) w.r.t. to K_{t+1} we obtain, for each time period *t*, the first-order condition

$$\beta^t v_2(K_t, K_{t+1}) + \beta^{t+1} v_1(K_{t+1}, K_{t+2}) = 0 \tag{9}$$

where v_1 and v_2 are the partial derivatives of the function *v* and where

$$v_1 = \left[1 + f'(K_{t+1})\right] u'(C_{t+1}) \tag{10}$$

and

$$v_2 = -u'(C_{t+1}). \tag{11}$$

We now divide (9) by β^t to obtain

$$\beta v_1 + v_2 = 0 \tag{12}$$

or, substituting (10) and (11) into (12)

$$\beta\left[1 + f'(K_{t+1})\right] u'(C_{t+1}) = u'(C_t). \tag{13}$$

Equation (13), together with equation (4)

$$K_{t+1} = K_t + f(K_t) - C_t \tag{4}$$

form a dynamical system that describes the evolution of the time paths for consumption and the capital stock. Given the initial condition (5), this system has a solution for each terminal value of the capital stock. The transversality condition (6) ensures that we pick, out of the many possible solutions, the optimum one.

To compute the steady-state, we eliminate the time subscripts from (13) and (4) to obtain, respectively

$$f'(K) = \frac{1-\beta}{\beta} \tag{14}$$

and

$$C = f(K). \tag{15}$$

Finally, to obtain the steady-state for the capital stock as in equation (14) in the Growth in Excel chapter, we substitute (2) into equation (14) above and solve for *K*, thus obtaining

$$K_{ss} = \left(\frac{1-\beta}{\beta\alpha\theta}\right)^{\frac{1}{\alpha-1}}. \tag{16}$$

References

Abel, Andrew B. (1975), "A Comparison of Three Control Algorithms to the Monetarist-Fiscalist Debate," *Annals of Economic and Social Measurement*, **4**, 239-252.

Adda, J. and R. Cooper (2003), *Dynamics Economics: Quantitative Methods and Applications*, The MIT Press, Cambridge, MA.

Aghion, P. and P. Howitt (1997), *Endogenous Growth Theory*, The MIT Press, Cambridge, MA.

Amman, Hans M. and David A. Kendrick (1996), "Forward Looking Variables in Deterministic Control", *Annals of Operations Research*, **68**, 141-159.

Amman, Hans and David A. Kendrick (1999a), *The Duali/Dualpc Software for Optimal Control Models: User's Guide*, Center for Applied Research in Economics, The Univ. of Texas, Austin, Texas, TP92-03 (revised December 1999), available at http://eco.utexas.edu/faculty/Kendrick.

Amman, Hans M. and David A. Kendrick (1999b), "Programming Languages in Economics", *Computational Economics*, **14**, 151-181

Amman, Hans M. and David A. Kendrick (1999c), "Linear Quadratic Optimization for Models with Rational Expectations", *Macroeconomic Dynamics,* (1999) **3**, 534-543.

Amman, Hans M. and David A. Kendrick (1999d), "Should Macroeconomic Policy Makers Consider Parameter Covariances?", *Computational Economics*, **14**, 263-267.

Amman, Hans M. and David A. Kendrick (2000), "Stochastic Policy Design in a Learning Environment with Rational Expectations", *Journal of Optimization Theory and Applications*, **105**, 509-520.

Amman, Hans M. and David A. Kendrick (2003), "Mitigation of the Lucas Critique with Stochastic Control Methods", *Journal of Economic Dynamics and Control,* **27**, 2035-2057.

Amman, Hans M., David A. Kendrick and John Rust (1996), *Handbook of Computational Economics*, Elsevier, Amsterdam.

Andersen, V. (1999), *Access 2000: The Complete Reference*, The McGraw-Hill Companies, New York.

Anderson, Gary and George Moore (1985), "A Linear Algebraic Procedure for Solving Linear Perfect Foresight Models", *Economic Letters*, **17**, 247-252.

Arrow, Kenneth and Frank Hahn (1971), *General Competitive Analysis*, Holden-Day, San Francisco.

Axelrod, Robert (1997), *The Complexity of Cooperation*, Princeton University Press, New Jersey

Axelrod, Robert and Leigh Tesfatsion (2004), "On-Line Guide for Newcomers to Agent-Based Modeling in the Social Sciences", at
http://www.econ.iastate.edu/tesfatsi/abmread.htm

Azariadis, C. (1993), *Intertemporal Macroeconomics*, Blackwell, Oxford, UK and Cambridge, MA, USA.

Barro, R. and X. Sala-i-Martin (1995), *Economic Growth*, The MIT Press, Cambridge, Mass.

Bauer, Richard J. (1994), *Genetic Algorithms and Investment Strategies*, John Wiley & Sons, Inc., New York.

Bellman, Richard (1957), *Dynamic Programming*, Princeton University Press, Princeton, N.J.

Beltratti, A, S. Margarita and P. Terna (1996,) *Neural Networks for Economic and Financial Modeling*, International Thompson Computer Press.

Bertsekas, D. (1995), *Dynamic Programming and Optimal Control*, Athena Scientific.

Binder, Michael and M. Hashem Pesaran (2000), "Solution of Finite Horizon Multivariate Linear Rational Expectations Models with Sparse Linear Systems", *Journal of Economic Dynamics and Control*, **24**, 325-346.

Blanchard, O and S. Fischer (1989), *Lectures on Macroeconomics*, The MIT Press, Cambridge, Massachusetts.

Blanchard, O. J. and C. M. Kahn (1980), "The Solution of Linear Difference Models under Rational Expectations", *Econometrica,* **48**, 1305-1311.

Blinder, A. (1997), "Distinguished Lecture on Economics in Government: What Central Bankers Could Learn from Academics - and Vice Versa", *Journal of Economic Perspectives*, **11**, Spring.

Blitzer, Charles R., Richard S. Eckaus, Supriya Lahiri and Alexander Meeraus (1992), "The Potential for Reducing Carbon Emissions from Increased Efficiency: A General Equilibrium Methodology", *Indian Economic Review*, **27**, 199-214.

Brandimarte, P. (2001), *Numerical Methods in Finance: a MATLAB-Based Introduction*, John Wiley & Sons, Inc., New York.

Brooke, Anthony, David Kendrick, Alexander Meeraus and Ramesh Raman (1998), *GAMS, A User's Guide*, available from the GAMS Development Corporation at http://www.gams.com.

Chakravarty, S. (1962), "Optimum Savings with a Finite Planning Horizon", *International Economic Review*, **3**, 338-355.

Chiang, A. (1984), *Fundamental Methods of Mathematical Economics*, Third Edition, McGraw-Hill.

Chiarella, Carl and Peter Flaschel (2000), *The Dynamics of Keynesian Monetary Growth: Macro Foundations*, Cambridge University Press, Cambridge, UK.

Chow, Gregory (1967), Multiplier, Accelerator, and Liquidity Preference in the Determination of National Income in the United States, *The Review of Economics and Statistics*, **XLIV**, 1-15.

Chow, Gregory (1973), Effect of Uncertainty on Optimal Control Policies, *International Economic Review*, **14**, 632-645.

Chow, Gregory (1975), *Analysis and Control of Dynamic Economic Systems*, John Wiley & Sons, New York.

Craine, Roger (1979), Optimal Monetary Policy with Uncertainty, *Journal of Economic Dynamics and Control,* **1**, 59-83.

Dantzig, G. B. (1963), *Linear Programming and Extensions*, Princeton University Press, Princeton, New Jersey.

Date, C. J. (1977), *An Introduction to Database Systems, 2nd ed.*, Addison-Wesley, Reading, MA.

Debreu, Gerard (1986), *Theory of Value: an Axiomatic Analysis of Economic Equilibrium*, Yale University Press, New Haven.

Deissenberg, Christophe (1987), "On the Minmax Lyapunov Stabilization of Uncertain Economies," *Journal of Economic Dynamics and Control*, **11**, 229-234.

Dervis, Kemal, Jaime de Melo and Sherman Robinson (1982), *General Equilibrium Models for Development Policy*, Cambridge University Press, Cambridge, UK.

Dickhaut, John and Todd Kaplan (1993), "A Program for Finding Nash Equilibrium", Ch. 7 in Varian (1993a).

Dixon, Peter, Brian Parmenter, John Sutton and D. P. Vincent (1982), *ORANI: A Multisectoral Model of the Australian Economy*, North Holland, Amsterdam.

Dixon, Peter and Brian Parmenter (1996) "Computable General Equilibrium Modelling for Policy Analysis and Forecasting", Chapter 1 in Amman, Kendrick and Rust (1996).

Dixon, Peter, Brian Parmenter, Alan Powell and Peter Wilcoxen (1992), *Notes and Problems in Applied General Equilibrium Analysis*, North Holland, Amsterdam.

Dorfman, Robert, Paul Samuelson and Robert Solow (1958), *Linear Programming and Economic Analysis*, McGraw-Hill Book Company, New York.

Duraiappah, Anantha K. (1993), *Global Warming and Economic Development*, Kluwer Academic Publishers, Dordrecht, The Netherlands.

Duraiappah, Anantha K. (2003), *Computational Models in the Economics of Environment and Development*, Kluwer Academic Publishers, Dordrecht, The Netherlands.

Dutt, Amitava Krishna and Jaime Ros (eds) (2003), *Development Economics and Structuralist Macroeconomics: Essays in Honor of Lance Taylor*, Edward Elgar Publishing, Cheltenham, Glos, UK.

Eckaus, Richard S. (1994), "Potential CO2 Emissions: Alternatives to the IPCC Projections" Mimeo M.I.T. Joint Program on the Science and Policy of Global Change.

Epstein, Joshua M. and Robert Axtell (1996), *Growing Artificial Societies: Social Science from the Bottom Up*, The MIT Press, Cambridge, Massachusetts.

Evanchik, Michael (1998), "Student Finance Model in Duali", term paper for Computational Economics course, Dept. of Economics, Univ. of Texas, Austin, Texas.

Fair, Ray C. and John B. Taylor (1983), "Solution and Maximum Likelihood Estimation of Dynamic Nonlinear Rational Expectations Models", *Econometrica*, **51**, 1169-1185.

Fisher, P. G., S. Holly and A. J. Hughes Hallett (1986), "Efficient Solution Techniques for Dynamic Nonlinear Rational Expectations Models", *Journal of Economic Dynamics and Control*, **10**, 139-145.

Francois , J. F. and C. R. Shiells (1994) (eds), *Modeling Trade Policy: Applied General Equilibrium Assessments of North American Free Trade,* Cambridge University Press, Cambridge, UK.

Froeb, Luke M. and Gregory J. Werden (1996), "Simulating the Effects of Mergers Among Noncooperative Oligopolists", Ch. 8, pp. 177-195 in Varian (1996).

Garson, G. D. (1998), *Neural Networks: An Introductory Guide for Social Scientists*, SAGE Publications.

Gibbons, Robert (1992), *Game Theory for Applied Economists*, Princeton University Press, Princeton, New Jersey.

Gilli, Manfred and P. Winker (2003), "A Global Optimization Heuristic for Estimating Agent Based Models", *Computational Statistics and Data Analysis*, **42**, 299-312.

Goffe, Bill (1996), "SIMANN: A Global Optimization Algorithm using Simulated Annealing", *Studies in Nonlinear Dynamics and Econometrics*, Vol. 1, Issue 3.

Goffe, Bill (2004), Resources for Economists on the Internet, a web site sponsored by the American Economics Association at http://www.rfe.org

Goldberg, D. (1989), *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Publishing Co., Reading, Mass.

Gomis, Pedro and A. Haro (2003), "Global Dynamics in Macroeconomics: An Overlapping Generations Example", *Journal of Economic Dynamics and Control*, **27**, 1941-1995.

Hall, Robert E. and John B. Taylor (1997), *Macroeconomics*, 5th edition, W. W. Norton & Company, New York.

Hansen, Lars Peter and Thomas J. Sargent (2001), *Elements of Robust Control and Filtering for Macroeconomics*, draft downloaded from Sargents web site at www.stanford.edu/~sargent, version of 23 March.

Herbert, Ric D. (1998), *Observers and Macroeconomic Systems*, Kluwer Academic Publishers, Dordrecht, The Netherlands.

Herbert, R. D. and R. D. Bell (1997), Visualization in the Simulation and Control of Economic Models, *Computational Economics*, **10**, 107-118.

Hicks, John R. (1937), "Mr. Keynes and the Classics: A Suggested Interpretation", *Econometrica*, **5**, 147-159.

Holly, S. and A. Hughes Hallett, (1989), *Optimal Control, Expectations and Uncertainty*, Cambridge University Press, Cambridge, UK.

Hughes Hallett, Andrew and Peter McAdam (eds*)* (1999), *Analysis in Macroeconomic Modelling*, Kluwer Academic Publishers, Boston and Dordrecht.

Johansen, Leif. (1960), *A Multi-Sectoral Model of Economic Growth*, North Holland Publishing Company, Amsterdam.

Jones, C. (1998) *Introduction to Economic Growth*, W. W. Norton & Company, New York.

Judd, Kenneth L. (1998), *Numerical Methods in Economics*, The MIT Press, Cambridge, MA

Judd, Kenneth L. and Leigh Tesfatsion (2005), *Handbook of Computational Economics II: Agent-Based Computational Economics*, Handbooks in Economics Series, North Holland, Amsterdam (forthcoming).

Juillard, Michel (1996), "DYNARE: A Program for the Resolution and Simulation of Dynamic Models with forward Variables through the Use of a Relaxation Algorithm", CEPREMAP Working Paper No. 9602, Paris.

Kendrick, David A. and Lance Taylor (1971), "Nonlinear Models for Economic Planning", Ch. 1 in Hollis B. Chenery (ed), *Studies in Development Planning*, Harvard University Press, Cambridge, Mass.

Kendrick, David A. (1981), *Stochastic Control for Economic Models*, McGraw-Hill Book Company, New York, see also Kendrick (2002).

Kendrick, David A. (1982a), "Caution and Probing in a Macroeconomic Model", *Journal of Economic Dynamics and Control*, **4**, 149-170.

Kendrick, David A. (1982b), "A Relational Database for the U.S. Economy", Ch. 3 in Charles P. Kindleberger and Guido di Tella*, Economics in the Long View, Essays in Honor of W. W. Rostow*, The MacMillian Press Ltd., London, **3**, pp. 63-82.

Kendrick, David A. (1990)*, Models for Analyzing Comparative Advantage*, Kluwer Academic Publishers, Dordrecht, The Netherlands.

Kendrick, David A. (1996), "Sectoral Economics", Ch. 6 in Amman, Kendrick and Rust (1996).

Kendrick, David A. (2002), *Stochastic Control for Economic Models*, Second Edition, available at http://www.eco.utexas.edu/faculty/Kendrick.

Kendrick, David A. (2005), "Stochastic Control for Economic Models: Past, Present and the Paths Ahead," *Journal of Economic Dynamics and Control*, **29**, 3-30.

Kendrick, David A., Alexander Meeraus and Jaime Alatorre (1984), *The Planning of Investment Activities in the Steel Industry*, The Johns Hopkins University Press, Baltimore, Maryland.

Kendrick, David A., P. Ruben Mercado and Hans M. Amman (2005), "Computational Economics: Help for the Underestimated Undergraduate", forthcoming in *Computational Economics*.

Keynes, John Maynard (1936), *The General Theory of Employment, Interest and Money*, Macmillan, London.

Kim, Seung-Rae (2004), "Uncertainty, Political Preferences, and Stabilization: Stochastic Control Using Dynamic CGE Models," *Computational Economics*, **24**, 97-116.

Koopmans, Tjalling (1951), *Activity Analysis of Production and Allocations*, Cowles Commission for Research in Economics, Monograph No. 13, John Wiley and Sons, New York.

Kozicki, Sharon and Peter A. Tinsley (2002), "Dynamic Specifications in Optimizing Trend Deviation Macro Models," *Journal of Economic Dynamics and Control*, **26**, 1585-1612.

LeBaron, Blake (2005), "Agent Based Computational Finance", forthcoming in Judd and Tesfatsion (2005).

LeBaron, Blake (2004), "Evolving Long Run Investors in a Short Run World", Keynote address to the Society of Computational Economics Conference, University of Amsterdam, The Netherlands, July.

Lee, Myong Hwal (1998), "Analysis of Optimal Macroeconomic Policy Design", Ph.D. Dissertation, Department of Economics, The University of Texas, Austin, Texas 78712.

Leontief, Wassily (1953), *Studies in the Structure of the American Economy*, Oxford University Press, Oxford, UK.

Letson, David (1992), "Simulation of a Two-Pollutant, Two-Season Pollution Offsets Systems for the Colorado River of Texas Below Austin," *Water Resources Research*, **28**, 1311-18.

Lin, Kuan Pin (2001), *Computational Econometrics: GAUSS Programming for Econometricians and Financial Analysts*, ETEXT Publishing, available at http://www.etext.net

Lofgren, H., R. Lee Harris and S. Robinson (2002), *A Standard Computable General Equilibrium (CGE) Model in GAMS*, Microcomputers in Policy Research 5, International Food Policy Institute, Washington D.C.

Malinvaud, Edmund (1977), *The Theory of Unemployment Reconsidered*, Basil Blackwell.

Manne, Alan S. and Richard G. Richels (1992), *Buying Greenhouse Insurance: The Economic Costs of CO2 Emission Limits*, The MIT Press, Cambridge, MA.

Markowitz, Harry (1952), "Portfolio Selection", *The Journal of Finance*, **7**, 77-91.

Marimon, Ramon and Andrew Scott (eds) (1999), *Computational Methods for the Study of Dynamic Economies*, Oxford University Press, Oxford, UK.

McCarl, Bruce A. and Uwe A. Schneider (2001), "Greenhouse Gas Mitigation in U.S. Agriculture and Forestry", *Science*, **294**, 2481-2482.

McKibbin, Warwick J. and Peter J. Wilcoxen (2002), *Climate Change Policy After Kyoto: Blueprint for a Realistic Approach*, Brookings Institution, Washington, D.C.

Mercado, P. Ruben, David A. Kendrick and Hans Amman (1998), "Teaching Macroeconomics with GAMS," *Computational Economics*, **12**, 125-149.

Mercado, P. Ruben and David A. Kendrick (1999), "Computational Methods for Macro Policy Analysis: Hall and Taylor's Model in Duali", in Hughes Hallett and McAdam (1999), Ch. 8, pp. 179-206.

Mercado, P. Ruben and David Kendrick (2000), "Caution in Macroeconomic Policy: Uncertainty and the Relative Intensity of Policy", *Economics Letters*, **68**, 37-41.

Mercado, P. Ruben, Lihui Lin and David A. Kendrick (2003), "Modeling Economic Growth with GAMS", Chapter 2, pp. 31-51 in Dutt and Ros (2003).

Mercado, P. Ruben (2004), "The Timing of Uncertainty and the Intensity of Policy", *Computational Economics*, **23**, 303-313.

Miranda, Mario J. and Paul L. Fackler (2002), *Applied Computational Economics and Finance*, The MIT Press, Cambridge, MA.

Nordhaus, William D. (1992), "An Optimal Transition Path for Controlling Greenhouse Gases", *Science*, **258**, 1315-1319.

Nordhaus, W. and J. Boyer (2000), *Warming the World: Economic Modeling of Global Warming*, The MIT Press, Cambridge, MA.

Nowak, Martin and Robert May (1992), "Evolutionary Games and Spatial Chaos", *Nature*, 359 (6398), **29**, 826-829.

Nowak, Martin and Robert May (1993), "The Spatial Dilemmas of Evolution", *International Journal of Bifurcation and Chaos*, **3**, 35-78.

Oudiz, G. and J. Sachs (1985), "International Policy Coordination in Dynamic Macroeconomic Models", in Willem H. Buiter and R. C. Marston, *International Economic Policy Coordination*, Cambridge University Press, Cambridge, England.

Paez, Pedro (1999), *An Optimal Control Framework for the Design of Alternative Macroeconomic Policies*, Ph.D. Dissertation, University of Texas, Austin, Texas.

Park, H. J. (1997), *A Control Theory Analysis of Macroeconomic Policy Coordination by the US, Japan and Korea*, Ph.D. Dissertation, Department of Economics, The University of Texas at Austin.

Passinetti, L. (1977), *Lectures on the Theory of Production*, Columbia University Press, New York.

Pesaran, M. H. (1987), *The Limits to Rational Expectations*, Basil Blackwell, Oxford, UK.

Pindyck, Robert S. (1973), *Optimal Planning for Economic Stabilization*, North-Holland, Amsterdam.

Rardin, R. (1998), *Optimization in Operations Research,* Prentice Hall, Englewood Cliffs, New Jersey.

Ricardo, David (1817), *Principles of Political Economy and Taxation*, Dover Books, http://store.yahoo.com/doverpublications. An alternative source for this book is given below.

Ricardo, David (1951), *Principles of Political Economy and Taxation*, in *The Works and Correspondence of David Ricardo*, Volume 1, Edited by P. Sraffa, Cambridge University Press for the Royal Economic Society, Cambridge, UK.

Roland-Holst, D. W., K. A. Reinert, and C. R. Shiells (1994), "A General Equilibrium Analysis of North American Economic Integration," in: Francois and Shiells (1994).

Ros, J. (2001), *Development Theory and the Economics of Growth*, University of Michigan Press, Ann Arbor, Michigan.

Rotemberg, J. and M. Woodford, (1997), "An Optimization-Based Econometric Model for the Evaluation of Monetary Policy," *NBER Macroeconomic Annual*, **12**, 297-346.

Rustem, Berc (1992), "A Constrained Min-Max Algorithm for Rival Models of the Same Economic System", *Math Programming*, **53**, 279-295.

Rustem, Berc and M. Howe (2002), *Algorithms for Worst-Case Design with Applications to Risk Management*, Princeton University Press, Princeton, New Jersey.

Sargent, T. (1987), *Dynamic Macroeconomic Theory*, Harvard University Press, Cambridge, MA.

Sargent, Thomas J. (1993), *Bounded Rationality in Macroeconomics*, Oxford University Press, Oxford, United Kingdom.

Sengupta, J. and P. Fanchon (1997), *Control Theory Methods in Economics*, Kluwer Academic Publishers, Boston.

Shoven, J. and J. Whalley, 1992, *Applying General Equilibrium*, Cambridge University Press, Cambridge, UK.

Shupp, F. (1976), "Uncertainty and Optimal Stabilization Policy", *Journal of Public Economics*, **6**, 243-253.

Silberberg, E. and W. Suen (2001), *The Structure of Economics: A Mathematical Analysis,* McGraw-Hill, New York.

Sims, Christopher A. (2002), "Solving Linear Rational Expectations Models", *Computational Economics*, **20**, 1-20.

Sraffa, P. (1972), *Production of Commodities by Means of Commodities*, Cambridge University Press, Cambridge, UK.

Stanford Encyclopedia of Philosophy (2005), entry on Evolutionary Game Theory, http://plato.stanford.edu/entries/game-evolutionary/

Stokey, N. and R. Lucas (1989), *Recursive Methods in Economic Dynamics*, Harvard University Press, Cambridge, MA.

Stone, J. R. N., (1961), *Input-Output National Accounts*, OECD, Paris.

Taylor, John B. (1993), *Macroeconomic Policy in a World Economy*, W. W. Norton & Company, New York.

Taylor, John B. (1998), *Monetary Policy Rules*, University of Chicago Press, Chicago.

Taylor, John B. and Harald Uhlig (1990), "Solving Nonlinear Stochastic Growth Models: A Comparison of Alternative Solution Methods", *Journal of Business and Economic Statistics*, **8**, 1-17.

Taylor, Lance (1990), *Socially Relevant Policy Analysis: Structuralist Computable General Equilibrium Models for the Developing World*, The MIT Press, Cambridge, MA.

Tesfatsion, L. (2005), "ACE: A Constructive Approach to Economic Theory," forthcoming in Judd and Tesfatsion (2005).

Thompson, Gerald L and Sten Thore (1992), *Computational Economics*, The Scientific Press, South San Francisco, CA.

Tinsley, Peter, Roger Craine, and Arthur Havenner (1974), "On NEREF Solutions of Macroeconomic Tracking Problems", 3d NBER Stochastic Control Conf., Washington..

Tucci, Marco (2002), "A Note on Global Optimization in Adaptive Control, Econometrics and Macroeconomics", *Journal of Economic Dynamics and Control*, **26**, 1739-1764

Turnovsky, Stephen (1975), "Optimal Choice of Monetary Instrument in a Linear Economic Model with Stochastic Coefficients", *Journal of Money, Credit and Banking*, **7**, 51-80.

Varian, Hal R. (1993a), *Economic and Financial Modeling with Mathematica*, Springer-Verlag, New York

Varian, Hal R. (1993b), *Intermediate Microeconomics: A Modern Approach,* W.W. Norton & Company, New York, New York.

Varian, Hal R. (1996), *Computational Economics and Finance: Modeling and Analysis with Mathematica*, Springer-Verlag Publishers, Santa Clara, California.

Wallis, K. (1980), "Econometric Implications of the Rational Expectations Hypothesis", *Econometrica*, **48**, no. 1.

Wolfram, Stephen (2003), *The Mathematica Book, 5th ed.,* Wolfram Media.

Woodford, Michael (2003), *Interest and Prices: Foundations of a Theory of Monetary Policy*, Princeton University Press, Princeton.

Zadrozny, Peter and Baoline Chen (1999), "Perturbation Solutions of Nonlinear Rational Expectations Models", presented at the Fifth International Conference of the Society of Computational Economics, Boston College, June 1999.