

# *Mathematica* as an Environment for Doing Economics and Econometrics

David A. Belsley<sup>1</sup>  
Boston College

June 11, 1997

Today's graduate students in economics must master early on a computational environment suitable for their research needs. A case is made here why *Mathematica* is an eminently reasonable choice for this purpose for many students. Salient features of *Mathematica* are examined in this context, and the breadth of economic research accomplished in *Mathematica* is illustrated.

## Introduction

Sooner or later, most of today's graduate students in economics will be faced with a critically important decision: which computational environment to learn and use.<sup>2</sup> And more often than not, they will be ill equipped to make this choice. While various of a student's professors and classmates will undoubtedly advocate specific software, such suggestions are often salted with confusing contradictions and unassessable evangelism, and, to make matters worse, local computer facilities will likely make only certain of the alternatives available at reasonable cost.

The need for this choice may possibly come to some as an unpleasant and belated surprise. Initially, course emphasis is likely placed on theory. But at some point, the rudeness of "doing" arises—sometimes not until well into the graduate curriculum, and sometimes not until the beginning of thesis writing. In years past, this problem might have been resolved simply by the student's learning to use a canned regression, statistical, or econometric package. But, more and more, students are finding the need to learn the programming skills to develop one's own models and statistical tools. And thus the daunting situation arises in which the student must undertake a major new learning experience totally distracting to any intended economic research.

Students' reactions to this vary. Some take time out to determine what software environment is needed and learn to use it well before returning to their economic researches. Some do the same, but get lost in the lure of their new-found game, never to return to economic research again. Others try to learn a software environment and simultaneously continue researches. Yet others compromise, mechanically learning "just what's needed" of particular software packages to keep their research momentum alive. A few simply

---

<sup>1</sup> The author is grateful to Christopher F. Baum for helpful comments.

<sup>2</sup> See Amman and Kendrick (1997) for a more complete discussion of this topic.

flounder. Except for the last, each of these reactions has its various pros and cons. However, I would argue that there are very few pros to the just-what's-needed approach. This method invariably results in little real understanding and produces skills that are quickly forgotten. When the skills are needed again or must be modified, it's "back to the drawing boards," often with an element of panic that can easily cause the student to block, i.e., resist undertaking important new research elements or even initiating the research at all. I would also argue that it typically requires an exceptional student simultaneously to learn a software environment and continue high-level graduate research.

Thus, the best solution to my mind is to begin at the very outset of one's graduate career learning, using, and becoming comfortable with some good computational environment—before it is truly needed and while there is time for the process to occur naturally and systematically, to become ingrained. The problem with this is that it requires the student to commit to a particular environment before knowing what may be optimal for his needs. This, however, I claim to be a small problem. For it is far better that the student have well in hand some (any) good environment that he knows and trusts and can use flexibly and comfortably in a multitude of needs as they arise than it is to wait to try to find a "best" environment.

And this is not a difficult thing to do; there are a variety of good packages that can serve this need. The virtue of this may well be seen in the fact that most established economic researchers believe that the environment that they learned and use is among the very best. People who were weaned on *GAUSS* cannot understand how anyone could be excited by *Maple*, while *Maple* users look askance at *Mathematica* users, etc. With so many happy and devoted users, each of these products must be doing something right. So the important thing is to pick one, learn it early and well, and use it often and productively—so that it becomes a real tool and not an impediment. This strategy assures that the student always has some suitable, if not optimal, computational facility at hand. And it has the added advantage that, having learned one environment, it is typically much easier to learn new ones. Thus, research can continue even if special elements of the research may benefit from the occasional learning of a more optimal procedure.

Numerous environments exist that will be of interest to graduate students in economics. These include such packages as *GAUSS*, *MATLAB*, *GAMS*, *Mathematica*, *Maple*, *Derive*, *Reduce*, *Macsyma*, *RATS*, *TROLL*, *PCGIVE*, *Ox*, *SAS*, *SPSS*, *S*, *S+*, and *Stata*. While each of these is potentially useful, they are not all the same type of environ. At the risk of oversimplifying, the first two, *GAUSS* and *MATLAB* are quite specially designed programming environments for doing matrix-type calculations easily and effectively. *GAMS* is also a special-purpose environment; it is designed for building and manipulating large-scale economic models. Similarly, the last nine are basically statistical packages, allowing one to do standard, and not so standard, statistical calculations with relative ease and flexibility. Among these, *RATS*, *TROLL*, and

*PCGIVE* are more econometrically oriented, and *S+* is oriented towards data analysis. *Stata* is somewhere between a statistical package and a fully extensible econometrics programming language. The remaining products, particularly *Mathematica*, *Macsyma*, and *Maple*, are more general environments that allow for a wide spectrum of operations, encompassing most of what has been mentioned, albeit perhaps at the cost of some efficiency in carrying out specific tasks. Among these, *Mathematica* and *Maple* are arguably the two most popular and widely used.

Thus, the strategy I suggest would point toward selecting an environment like *Mathematica* or *Maple*. These provide powerful modeling, graphical, statistical, and programming facilities that allow the user, within a single environment, to do almost everything. And even if more special-purpose routines are eventually called for, they allow for rapid prototyping and testing of projects that makes subsequent implementation in other environments very much easier and surer.

Both *Mathematica* and *Maple*, for very good reasons, have their devoted adherents. I happen to be a *Mathematica* sort, and it is *Mathematica* that I want to deal with here. It is not, however, my purpose to “sell” *Mathematica*. Rather I want to try to elucidate for those who are considering computational environments what *Mathematica*’s pros and cons are for use in economic research. To this end, I will try to explain some of *Mathematica*’s salient features and illustrate the kinds of economic research that have successfully been accomplished using *Mathematica*.

This latter objective is made easier by the presence of two excellent volumes, *Economic and Financial Modeling with Mathematica* and *Computational Economics and Finance: Modeling and Analysis with Mathematica*, both edited by Hal Varian. Typically, scholarly papers in economics publish methods and results but provide little insight into the means by which those results are produced. Indeed, one is usually not made privy to the computational environment used. The studies in these volumes, however, display both means and results in full. Those whose interest in *Mathematica* is piqued here are well advised to examine these volumes in detail, for they give an excellent idea of the power of *Mathematica* for doing economic/econometric research in a wide variety of fields and provide good examples of how it is done.

## Basics

My purpose here is not to provide a tutorial in *Mathematica*. For this, there is little that surpasses the introduction of Stephen Wolfram’s book *Mathematica*, now in its third edition, which also contains complete *Mathematica* documentation. This book is available separately from the software and is absolutely essential. The curious with access to *Mathematica* are well advised to borrow or buy this book and “do” chapter 1. It’s a short period of time well spent. Rather I wish first to give the reader a quick feel for and appreciation

of what *Mathematica* is and what it can do, and then we can look at its uses in economics.

*Mathematica* is a fully integrated symbolic algebraic computational environment. By fully integrated I mean that, within a single environment, one can do calculations, algebra, calculus, and matrix manipulations; one can solve equations, build models, program, enact rule-based transformations, and one can graph the numeric results. By symbolic, I mean that most of these operations can take place either on numbers (numerically) or on abstract symbols (symbolically), a feature that is particularly useful in economics. The heart of this integration lies in the simple, but often overlooked, fact that, in *Mathematica*, everything—and I mean everything—is a list. The more you use *Mathematica*, the more you will realize that it is this commonality of structure that gives *Mathematica* its unity, form, and power. And once this fact is truly appreciated, one has the basis for really understanding how to use *Mathematica* and how best to exploit its almost endless possibilities.

A common mistake in learning an environment like *Mathematica* is trying to learn it all at once. *Mathematica* is a gargantuan program, containing far more substance than you will ever need or want. Most economists, for example, will likely never need Bessel or Möbius Mu functions. But *Mathematica* was not written just for economists, so you must learn wisely. At first, things are easy; a good deal of *Mathematica* can be used, even productively, without great effort. You can solve equations, do calculus, draw graphs, manipulate matrices almost immediately using *Mathematica*'s many built-in features. But more ambitious projects require one's own programming. And here *Mathematica* excels.

*Mathematica*'s programming language is extensive, with commands borrowed from virtually every known language: LISP, FORTRAN, C, Pascal, Algol—you name it. Thus, those coming to *Mathematica* with programming experience will likely be at home with elements of *Mathematica*'s language and will be able to use many of their familiar programming practices. But beyond this, *Mathematica* has its own functional programming structure that, for many purposes, is so much more elegant and powerful that you will find yourself slowly gravitating away from the familiar and into the new. The problem is that this transition involves a slow, long, and virtually endless learning curve. But this is not a cause for despair—indeed, quite the opposite. It simply means that the more you use *Mathematica*, the more you will be able to use it and do with it.

*Mathematica*, then, is an environment that one can start using quickly, easily, and productively for many standard needs. But it is also an environment that can, with learning and effort, accommodate almost any computational research need.

## An example

Let's give a simple example of a problem every first-year grad student knows to be easy in concept, hard in practice: dealing with determinants of  $4 \times 4$  matrices. Recall, everything in *Mathematica* is a list, and the basic list in *Mathematica* is simply a set of objects, separated by commas within braces. Thus a 4-vector of numbers is simply a list like

```
In[ ]:=  
  {-1, 2, 4, 2}
```

and a  $4 \times 4$  matrix is simply a list four such lists, such as

```
In[ ]:=  
  {{1, a, 6, 5}, {-1, 2, 4, 2}, {5b, 6, b, 4}, {-1, -2, 2, 4}}
```

We note that some of the elements here are numbers and some are symbols involving **a** and **b**. If I'm interested in this matrix for different values of **a** and **b**, I can write

```
In[ ]:=  
  mat[a_, b_] := {{1, a, 6, 5}, {-1, 2, 4, 2}, {5b, 6, b, 4}, {-1, -2, 2, 4}}
```

to create a matrix **mat** as a function of **a** and **b**. Thus, the one gets the following input and result for the determinant of **mat** for **a** equal to 6 and arbitrary **b**:

```
In[ ]:=  
  Det[mat[6, b]]
```

```
Out[ ]:=  
  -300 + 344b
```

And, if one wants to know the value of **a** for which the determinant of **mat** is zero, one simply gives and receives

```
In[ ]:=  
  Solve[Det[mat[a, b]] == 0, a]
```

```
Out[ ]:=  
  a ->  $\frac{2(87 + 7b)}{4 + 31b}$ 
```

Recall that **a** and **b** are being used symbolically here. This expression holds for any **b**. Impressed? I hope so; otherwise do it out by hand! But realize that this is a trivial use of *Mathematica*. Imagine, then, a non-trivial use. Slightly higher on the scale, for example, would be to find the eigenvalues of **mat** when **a** is, say, 40 and **b** is 12; i.e.,

```
In[ ]:=  
  Eigenvalues[mat[40, 12]] // N
```

Out]:=

```
{-8.71359 - 14.5496 I, -8.71359 + 14.5496 I, 3.06334, 33.3639}
```

One could even ask for, and get

In]:=

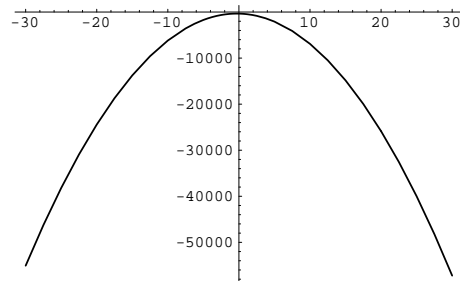
```
Eigenvalues[mat[a,b]]
```

i.e., the eigenvalues of `mat` for general `a` and `b`, but the output is too long and messy, even after simplification, to print here (but, if you've got *Mathematica*, you can do it for yourself).

Now, suppose we were interested in seeing how the determinant of `mat` changes when `a = -b` over the range of -30 to 30; then

In]:=

```
Plot[Det[mat[-a,a]], {a, -30,30}]
```



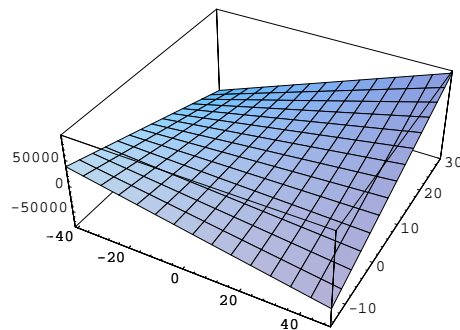
Out]:=

```
-Graphics-
```

or, more interestingly yet, a 3-dimensional graphic for separate values of `a` and `b`:

In]:=

```
Plot3D[Det[mat[a,b]], {a,-41,50}, {b,-15,30}]
```



Out]:=

```
-Graphics-
```

Here we see how graphics can be produced seamlessly within the *Mathematica* environment. But even with this introduction, I suspect you will be amazed at how quickly and effortlessly the authors of the papers in the two volumes cited above are able to produce graphical illustrations of their results.

### **The structure of *Mathematica***

*Mathematica* is available for almost all popular platforms, including Macintosh, NeXT, and IBM personal computers, and Unix workstations. It comes in two parts, the front-end and the kernel. These can either be run together on the same machine or separately on different machines. The kernel does the actual computational crunching and is, in principle, standardized across machines. The front-end is written to take advantage of the machine it is run on. On personal computers like the Mac and IBM, the front-end takes the form of a notebook. Notebooks must be seen to be fully understood and appreciated, but they provide one of the most flexible and useful interactive working environments imaginable. They are unique to the *Mathematica* environment.

The notebook is an excellent word processor that also interacts with the kernel. All of *Mathematica*'s input and output are recorded in a notebook—along with anything else you wish to include. Thus, using notebooks, one can experiment, make comments, separate wheat from chaff, draft and organize sections, hide and display material at will in an easily formatted sectional hierarchy. This makes it possible to document, draft, and even write up your research as you go. It is also possible to print final manuscripts directly out of notebooks, and books can be produced from them. Indeed, many of the chapters in the Varian volumes cited above were constructed as notebooks. Version 3.0 of *Mathematica*, released at the beginning of this year, allows for 2-dimensional mathematical notation (the way mathematics are typically printed in books) with a TeX-like, or Scientific Word-like, input. It is now quite possible to print a final book or paper directly out of a *Mathematica* notebook.

One can link a notebook front-end on one machine to a kernel running remotely on another. Thus, one can take advantage of the superior graphics and versatility of a Macintosh for one's input and output while running the kernel remotely on some other very fast machine, which can be useful for very large projects. However, many of today's desktops have adequate memory and power to accommodate most *Mathematica* projects running barefoot. *Mathematica* can also make use of **MathLink** to interact with numerous other familiar programs such as Excel.

### ***Mathematica*'s language**

*Mathematica*'s programming language cannot be described in a few paragraphs—or even a few books. But some interesting notions can be introduced here that relate to the type of research economists do. Although

the reader may not be able to appreciate them immediately in full, they should germinate with the reader's increased knowledge of *Mathematica*. First, it is worth noting that *Mathematica* contains close variants of all the standard iterative and logical (branching) commands of other well-known languages, *Do*'s, *For*'s, *If*'s, etc. But it is what is beyond this that I wish to introduce now.

A general object in *Mathematica* is a list of items  $e_1$  to  $e_n$  with a head, written

```
obj = Head[e1,...,en].
```

Virtually everything in *Mathematica* has this form; the vector we used above is simply a list of items with head `List`

```
list = List[e1,...,en] = {e1,...,en},
```

which can also be written in shorthand using braces as noted. This much is easily understood, but, as I said, everything in *Mathematica* is a list. Indeed, consider the operation of addition. This is accomplished with the head `Plus` acting on the list of things to be summed. Thus,  $\sum_{i=1}^n e_i$  becomes

```
sum = Plus[e1,...,en] = e1+...+en.
```

While *Mathematica* allows the conventional notation  $e_1 + \dots + e_n$  as an obvious convenience, it is translated internally into the `Plus[]` form. Likewise, a function is a list with head `Function` and elements that describe its operation. So are the standard logical forms—the familiar “if” statement is a list with head `If` and objects telling what to do in the true and false cases:

```
If[expression, true, false, unknown].
```

It is interesting to note that, in *Mathematica*, `true` and `false` need not be the only possible outcomes. The evaluation of an `expression`, such as `a > 1`, branches to the `true` or `false` procedures only if *Mathematica* knows for sure whether `a > 1`. Recall that *Mathematica* can deal with symbols, so if `a` has not actually been assigned a numerical value, *Mathematica* has no way of knowing if `a > 1`, and hence it branches to the procedure for `unknown`. [Standard Aristotelian behavior can be accomplished by wrapping `expression` in `TrueQ[]`, which is a function that returns `false` unless `expression` is true.]

Because of this commonality of form, all objects in *Mathematica* can be treated in similar ways, a fact that allows for *functional programming*, a concept that we will develop somewhat as we proceed. Consider, then, the built-in *Mathematica* function `Map[]`, an object whose power will impress you more the more you use it. `Map[]` distributes a function `f` over the elements of an object. For a general object with head `Head`, we have

```
Map[f,Head[e1,...,en]] -> Head[f[e1],...,f[en]].
```

If the object is a `List`, we have

```
Map[f,{e1,...,en}] -> {f[e1],...,f[en]}
```



i.e., mapping `f` on a list creates a list of `f`'s. And if the object is `Plus`, we have

```
Map[f,Plus[e1,...,en]] -> f[e1]+...+f[en].
```

`Map`, then, allows operations on each element of an object without having to refer to each element. It is, in fact, an automatic *Do* or *For* statement that requires no indexing, no looping, and no other management. This is one aspect of functional programming, namely, blocks of usual programming code that require management are incorporated into a single functional statement that requires no management. In *C*, such code might minimally take the form

```
long e[30], g[30];
void myproc(void)
{
    short i;

    for(i=1; i<=30; i++)
        g[i] = f(e[i]);
}
```

You must manage everything here: defining the arrays, declaring them, setting and specifying the limits, and defining the procedure to accomplish the task. And when you are through, unless additional code and memory management are provided to allow dynamic sizing, your code works only for an array of size 30 of long numbers. All this, however, is subsumed in the single `Map[ ]` statement. It does the typing, the limit setting, the incrementing, the looping, and the determination of the size of the array, whatever that size may be and whatever its objects may be. Indeed, in *Mathematica*, almost all instances of iterative code can be replaced with the `Map[ ]` function or its allies. The result is simpler code that is more readily and generally implemented and more quickly executed.

Another important example of such functional programming is the `Apply[ ]` function. Since all objects are lists with heads, one can change one kind of object to another merely by changing its head—and this is what `Apply[ ]` does. Thus

```
Apply[Newhead, Head[e1,...,en]] -> Newhead[e1,...,en].
```

So, for example, one can change a sum to a product with

```
Apply[Times, e1+...+en] -> e1*...*en.
```

Again we have a procedure with great power. Suppose, as an example, one wants the product of the logarithms of the elements of a data vector `vec` (of whatever length it happens to be). Then write

```
ln[]:=
Apply[Times,Map[Log,vec]]
```

or, more simply,

In[ ]:=

```
Apply[Times, Log[vec]]
```

since *Mathematica* converts vector and matrix shorthand statements like `Log[vec]` into their obvious mappings.

The `Thread[ ]` function, which distributes a function `f` over several lists in parallel, provides yet another important example of functional programming. Here, we have

```
Thread[f[{a1,...,an},{b1,...,bn}]] -> {f[a1,b1],...,f[an,bn]}.
```

And if the potential of this is not immediately obvious, stop and juggle it about in your mind for a while.

Next, we note that *Mathematica* allows for “pure” functions: simply adding an `&` to the end of a *Mathematica* expression `Expr`—as in `Expr&`—causes it to become a function for doing `Expr`. Thus, if one has an expression like `x+10*y>0.0`, this can become a function returning `True` and `False` for arguments `#1` & `#2` simply by writing `(#1+10*#2>0.0)&`. And if one now defines

In[ ]:=

```
f = (#1+10*#2>=0.0)&
```

then the expression `f[a,b]` works just like a regular function with arguments `a` and `b`. One simply cannot imagine the power and flexibility that such pure functions afford, particularly when combined with such functional constructs as `Map[ ]` and `Apply[ ]`.

Finally, in this short list of *Mathematica* power tools that are of great use to the types of research that economists do, we note that *Mathematica* allows for hierarchical function definitions; that is, the same function name can be given many different definitions. This makes it possible to write very complex functions very easily. To provide a simple illustration of this, suppose one wishes to express the idea that an economic relation is quadratic for positive `x` but linear with slope 3 for negative `x` and with margins that match at `x=0`. One need only write

In[ ]:=

```
f[x_] := 3*x /; x < 0.0  
f[x_] := 3*x+x^2 /; x >= 0.0
```

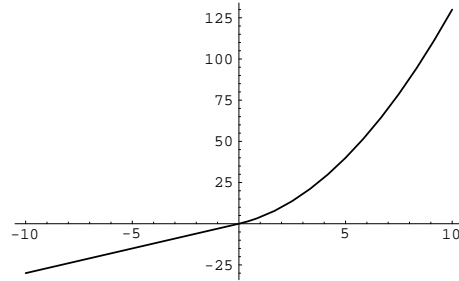
The `/;` operator provides a condition, so that the first definition applies only when `x < 0`, whereas the second applies when `x ≥ 0`. But now `f` can be treated as a single function; thus

In[ ]:=

```
Plot[f[x], {x, -10, 10}]
```

Out[ ]:=

```
-Graphics-
```



### The Cons of *Mathematica*

The foregoing would surely seem to indicate *Mathematica* to be a seductive environment for doing economic/econometric research, and it certainly has a great deal to be said for it. But it also has its share of cons. *Mathematica* has a long learning curve. While many of *Mathematica*'s powerful features can be used immediately by the novice to great effect, the full power of its programming language comes only with time, energy, and experience. This will be true of any worthwhile research environment. In the long-run, however, the learning will be worth the trouble. That is why I suggest at the beginning of this piece that the student commit early to a good environment, and use it well and often. Use it to do even simple things for which it is overkill. The more one uses a program like *Mathematica*, the more useful it becomes.

*Mathematica* is also a real memory hog. Part of the reason it is able to relieve the user of having to specify types for numbers (such as `int`, `long`, `short`, `extended`, `real`, etc.) is because it simply holds on to everything in some appropriate high-precision form. Arrays of a million numbers that can be made to fit in moderate core sizes with 8 digit precision can readily blow all bounds when kept with 19 or 20 digit precision. Furthermore, *Mathematica* retains all of the input and output of a given session along with added material for screen display. This can be extremely useful, allowing prior results to be recalled at will, but it can also chew up memory unmercifully.

The symbolic nature of *Mathematica* is one of its delights, but it is also able to produce unexpected and unpleasant surprises. One must use great care to avoid, for example, inadvertently defining an object being used as a symbol. If  $x$  is being used symbolically, and somewhere it is written `x = 3`, then  $x$  will be replaced in every subsequent use by 3—great if it's intended, but devastating if not. One safeguard is to use the `Clear[]` command before any such use to be sure the slate has been wiped clean. The power and complexity of *Mathematica* allows it to produce many other sorts of surprises as well, but one gets cleverer at anticipating and avoiding such things as one gains experience.

*Mathematica* also runs user code only in interpreted mode; that is, it examines and compiles each statement as it encounters it. There is no precompiling with its attendant increase in run-time speed and efficiency. This can be a real advantage while developing code but can become a source of concern during

crunch time. There is a `Compile[]` function that can be used in some situations to great effect, but a more general compilation facility would be a lovely addition to this environment. *Mathematica*'s own built-in functions, of course, are already compiled and run very quickly; they should be used whenever possible. The `Map[]` function, for example, runs as compiled code whereas an equivalent user defined do-loop would run in a much slower interpreted mode. Thus, one encounters the somewhat ironic situation that a home-brewed Cholesky decomposition, a routine often used because of its computational efficiency, will run much more slowly than the more general *Mathematica* built-in function `SingularValues[]` that carries out the computationally much more involved and demanding singular-value decomposition.

Finally, *Mathematica* is very expensive. Indeed, one of my Italian associates vouchsafed that, while *Mathematica* is a wonderful language and environment, it is so expensive that it is difficult to justify to one's administration to buy such a program for economics; perhaps if it were called *Economica*, he suggested. Well, it could be so called—or *Physica*, or *Chemica*, or *Psychologica*, or *Statistica*. I hope by now we have seen that *Mathematica* is a chameleon that can readily take a multitude of hues depending upon need. However, *Mathematica*'s academic pricing is really quite reasonable, particularly considering that it buys the entire package. Several rival products cost as much or more only for the basics; important functions that are part of *Mathematica* must be purchased separately at considerable additional cost. And *Mathematica*'s expense can be even further ameliorated once a university acquires a site license. For then its organizational use to members of that community is free, and private copies become available to those members at an extremely reasonable and competitive cost. Such a site license is truly a social good.

### ***Mathematica* and Economic Research**

*Mathematica* can be, and has been, used to do virtually any kind of economic research. Let's look at some examples. As noted, these examples come from the two volumes *Economic and Financial Modeling with Mathematica* and *Computational Economics and Finance*, edited by Hal Varian. To make references easier, Chapter  $n$  of the first and second volumes will be denoted I. $n$  and II. $n$ , respectively.

**Theory:** All first-year microtheory courses do the theory of the consumer and producer, many using Varian's textbook. If you would like to see some of this material done symbolically in *Mathematica*, look at Hal Varian's piece I.1, *Symbolic Optimization*. Unconstrained and constrained optimization are dealt with along with tests for definiteness, and examples are given of comparative statics, Cobb-Douglas and CES utility and cost functions, and monopoly behavior. Also exemplified are numeric and symbolic dynamic programming, both stochastic and non stochastic. In *Efficiency in Production and Consumption*, II.6, Varian uses *Mathematica* to develop measures of efficiency, the degree to which actual production or consumption

behavior achieves optimality.

General equilibrium models, with their dense notation and potentially large sizes, are always a challenge. In *General Equilibrium Models*, I.5, Asahi Noguchi shows how to build general equilibrium models in *Mathematica*. Two and three sector/factor models are analyzed and graphed. The framework is ripe for expansion.

*Mathematica*'s pattern recognition facility, by the way, provides an excellent framework in which to do model building, both for economic theory and econometrics. I have yet to see it fully exploited in such dimensions.

**Optimization:** Optimization is a very important element in much of economic analysis, and, as you might expect, *Mathematica* provides a very powerful and flexible environment both for carrying out optimization techniques and for developing appropriate optimization algorithms.

*Mathematica* has its own Linear Programming package `LinearProgramming.m` that can be of great service to many economic applications. A *Mathematica* "package" is essentially a specialized notebook providing a routine or a collection of related routines written for *Mathematica*. Hundreds of such packages are available, either as part of *Mathematica* from MathSource, made available by Wolfram Research (the makers of *Mathematica*), or from *Mathematica* usergroups, or written by the user. They are typically identifiable by a `.m` appended to their names.

Those interested in learning about Linear Programming, or those interested in learning about the use of *Mathematica* as an medium for teaching such topics as Linear Programming, will profit from Michael Carter's *Linear Programming with Mathematica*, II.1. In addition to a lucid explanation of the topic and the basic simplex method for its solution, additional Linear Programming packages are provided. This package supplements the one provided with *Mathematica* and is explained more fully in II.2, *Linear Programming with Mathematica: Sensitivity Analysis*, also by Michael Carter. Here more advanced uses of LP are given.

J.-C. Culioli's work, *Optimization with Mathematica*, II.3, provides even more general and powerful means for optimization within *Mathematica*. In addition to a demonstration of how to use (and how not to use) *Mathematica*'s built-in general procedure `FindMinimum[ ]` for doing unconstrained nonlinear programming, the package `MultiplierMethod.m` is provided for accomplishing a wide variety of more complex optimization problems, including nonlinear programming with nonlinear equality and inequality constraints. Thus, consider the problem

$$\min y + x^2 \text{ s.t. } x + y = 3.5; -x + 2 \leq 0, -y + 1 \leq 0,$$

whose solution is found as simply as

```
In[]:=
MultiplierMethod[y+x^2, {x+y-3.5}, {-x+2,-y+1}, {x,y}
{4.,6.}, DualParameter -> True]
```

```
Out[]:=
{5.5, {x -> 2., y-> 1.5}, {-4.44089 10-16}, {MMethod'p1 -> -1.},
{2.48065 10-8, -0.5}, {MMethod'q1 -> 3., MMethod'q2 -> 0.}}
```

Here, in addition to the value of the objection function 5.5, we have a solution for  $\mathbf{x}$  and  $\mathbf{y}$ , a check that the equalities and inequalities are satisfied, the value for the Lagrange multiplier  $\mathbf{p1}$  for the equality, and the values for the dual Kuhn-Tucker parameters  $\mathbf{q1}$ , and  $\mathbf{q2}$ . Many other examples are given in the Culioli study.

Further forms of optimization techniques are found in Paul Rubin's *Optimizing with Piecewise Smooth Functions*, II.4. From the short description given above of *Mathematica*'s facility for defining segmented functions, you can well imagine that it is a natural environment for this purpose. Such piecewise functions occur quite naturally in many inventory and production problems, and are being used more and more in various nonparametric statistical procedures.

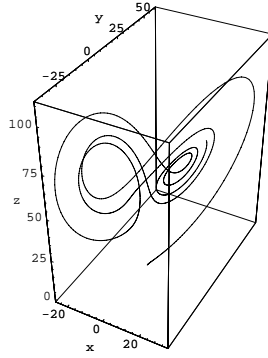
**Dynamics and Growth:** An excellent exposition of *Mathematica*'s abilities to handle numerous dynamical problems of value to economics is to be found in John Eckalbar's piece, I.3, *Economic Dynamics*. Means for dealing with one, two, and three variable systems, both linear and nonlinear, are shown along with elegant use of the graphics to demonstrate paths to equilibrium and relevant force fields. Consider, for example, solving and plotting the dynamic path of the following nonlinear 3-variable differential system:

$$\begin{aligned}x' &= -10x + 10y \\y' &= -xz + 60x - y \\z' &= xy - (8/3)z.\end{aligned}$$

```
In[]:=
sol2=NDSolve[{x'[t]==-10x[t]+10y[t],
y'[t]==-x[t]z[t]+60x[t]-y[t],
z'[t]==x[t]y[t]-(8/3)z[t],x[0]==1,
y[0]==1,z[0]==1}, {x,y,z}, {t,5}]
```

```
In[]:=
ParametricPlot3D[Evaluate[{x[t],y[t],z[t]}/.sol2], {t,0,3.3},
PlotPoints->1000, PlotRange->All, AxesLabel->{"x","y","z"}]
```

```
Out[]:=
-Graphics-
```



Gary Anderson, in *Symbolic Algebra Programming for Analyzing the Long Run Dynamics of Economic Models*, I.6, exploits *Mathematica*'s symbolic nature to analyze various nonlinear economic dynamic systems including a money-demand model, McCallum's real business-cycle model, and an overlapping-generations model.

Growth models can also be analyzed with *Mathematica*. Kenneth L. Judd and Sy-Ming Guu, I.4, in *Perturbation Solution Methods for Economic Growth Models*, make use of perturbation-solution methods to find optimal growth solutions (or good approximations thereto). They examine both continuous and discrete time models, and both stochastic and nonstochastic formulations.

**Industrial Organization:** Industrial Organization today is suffused with game theory, and it appears that *Mathematica* is a wonderful environment for gaming. A crash course in game theory with exemplar code for determining the various Nash equilibria for a two-person, zero-sum game is given by John Dickhaut and Todd Kaplan, I.7, in *A Program for Finding Nash Equilibria*. Here *Mathematica* is used to illustrate familiar and common economic tools as well as the Prisoner's Dilemma and the Battle of the Sexes. Todd Kaplan and Arijit Mukerji, I.2, use game-theoretic techniques based on the Bayesian-Nash equilibrium in the context of *Designing an Incentive-Compatible Contract*. In addition to a good graphical depiction of single-crossing utility functions, this work illustrates *Mathematica*'s graphics and equation-solving facilities as well as how a *Mathematica* session is easily expanded to deal with ever more complex cases—a two-person game grows to finite and, ultimately, limitingly large numbers. A Kuhn-Tucker package is also offered for more complex solutions. Michael Carter, I.8, uses *Mathematica* to represent and provide solutions for various *Cooperative Games*. This is a good environment for formulating, solving and analyzing games and useful graphical intuition is provided.

Game theory is also used in *Mathematica* by William W. Sharkey, II.7, in *Cost Allocation* along with other techniques to implement cost-allocation pricing rules in both discrete and continuous situations. Means are also provided for solving for optimal pricing rules when consumer demand functions are given. And

Luke M. Froeb and Gregory J. Werden, II.8, in *Simulating the Effects of Mergers Among Noncooperative Oligopolists*, overcome shortcomings of standard techniques for predicting the effects of mergers by using *Mathematica* to simulate the merger process using standard oligopoly models.

**Finance:** Options-pricing methods have figured prominently of late in the financial literature—the Black-Scholes method in particular. Ross M. Miller, I.12, in *Option Valuation* implements the Black-Scholes method in *Mathematica* as well as a symbolic analysis of its delta. This piece also illustrates how object-oriented approaches can be accommodated within *Mathematica*. Of course, the closed-form solution of the Black-Scholes method cannot be used to price all types of options. So Simon Benninga, Raz Steinmetz, and John Stroughair, II.11, in *Implementing Numerical Option Pricing Models*, show how to use *Mathematica* to price options using numerical methods such as the binary and binomial options-pricing models and by Monte Carlo techniques.

High frequency financial data often possess strong autocorrelation whose variance becomes a centerpiece of the of analysis. Luke M. Froeb, II.13, in *Log Spectral Analysis* presents a *Mathematica* package, `LogSpec.m`, for conducting a spectral analysis of such data and applies it to an analysis of the futures/spot price relationship and to test restrictions using the S&P 500 spot and futures data. This is a good example of using *Mathematica* to develop time-series techniques.

Auctions have also been studied within *Mathematica*. In *Auctions*, John Dickhaut, Steve Gjerstad, and Arijit Mukherji, II.9, examine and represent various first- and second-price auction mechanisms. Their behavior is also graphed.

The complex pricing schemes adopted by many airlines are examples of yield management. Ward Hanson, II.10, in *Yield Management* shows how to use *Mathematica* to analyze and implement such pricing schemes. This piece is also an excellent example of how to use the computational kernel of *Mathematica* interactively with a spread sheet like *Excel* as a “user friendly” front end.

In a piece *YieldCurve* that exemplifies a balanced spectrum of *Mathematica* facilities, Mark Fisher and David Zervos, II.12, provide a *Mathematica* package of the same name that extracts the term structure of interest rates from a set of default-free, pdv-priced bonds.

**Statistics/Econometrics:** As noted at the outset, *Mathematica* is a general-purpose environment, and, as such, it is inefficient to try to make it behave like a special-purpose statistical package. But this in no way means that *Mathematica* is not a wonderful environment in which to do statistical and econometric things. *Mathematica* does indeed have a number of very important statistical items. A set of basic statistics packages exists that defines a full panoply of statistical objects. And *Mathematica*’s built-in matrix language



and routines provide basic building blocks for virtually all statistical needs. Further, standard packages exist defining most distribution functions (and their inverses), and an excellent random number generator is evoked with `Random[]`. So, one can readily create within *Mathematica* statistical objects and procedures of all descriptions, including those that are difficult to find elsewhere. And, if need be, one can recreate standard procedures specialized to one's personal requirements.

Thus, in *Bounded & Unbounded Stochastic Processes*, Colin Rose, I.11, builds on *Mathematica*'s `Statistics.m` package to produce relevant statistics and distributions for bounded and truncated random variables. This allows one to deal more realistically with economic variables like prices or interest rates that are bounded from below. In this work, one finds routines for determining the means, variances, densities and cumulative distributions for bounded/truncated random variables. Examples are given using these notions for exchange rate target zones (stipulated bounds for exchange-rate variation) and irreversible investment under uncertainty.

In another vein, J. Michael Steele and Robert A. Stine, I.9, in *Mathematica and Diffusions*, review the notion of diffusions in economic time series and develop means to define and represent them through stochastic calculus in *Mathematica*. These techniques are then applied within the Black-Scholes options-pricing formula. A stochastic calculus package is also developed that includes a multivariate version of Itô's formula. These concepts are given greater generality and refinement in Wilfrid S. Kendall's, I.10, piece, *Itovsn3: Doing Stochastic Calculus with Mathematica*.

*Mathematica*'s ability to deal symbolically would seem ideal for optimization techniques that use symbolic differentiation. Stephen J. Brown, I.13, in *Nonlinear Systems Estimation: Asset Pricing Model Application* shows how to use the symbolic differentiation facility in *Mathematica* to give analytic expressions for the derivatives needed by nonlinear estimation packages.

And Eduardo Ley and Mark F. J. Steel, I.15, in *Bayesian Econometrics*, provide an introductory package that does the better part of Zellner's classical Bayesian analysis (sorry, I couldn't resist the pun). It is interesting to see how these techniques can be implemented with *Mathematica*, again with graphical embellishments. And it is amazing to see how paragraph built upon paragraph can suddenly be expressed in *Mathematica*'s powerful language in several well-chosen expressions. An application of Bayesian analysis to decision theory is given by Robert J. Korsan, I.17, in *Decision Analytica: An Example of Bayesian Inference and Decision Theory Using Mathematica*. This piece also provides a package for doing influence analysis.

David A. Belsley, I.14, in *Econometrics.m: A Package for Doing Econometrics in Mathematica* provides an econometric package (and documentation) for conducting Ordinary Least Squares, Two-Stage Least Squares, Instrumental Variables, and Mixed-Estimation as well as conditioning diagnostics. This procedure

interacts seamlessly with *Mathematica*'s other facilities (such as transformations and graphics) to provide a package of significant power and flexibility, allowing one to do many econometric procedures without recourse to additional software.

*Mathematica* also is a wonderful environment in which to develop Monte Carlo experiments. Those who have attempted such studies in other environments will not believe how quickly an experiment can be designed and set underway in *Mathematica*. This topic is dealt with in *Doing Monte Carlo Studies in Mathematica* by David A. Belsley, II.15. *Mathematica*'s excellent random-number generator is discussed here as well.

The means for handling standard (ARIMA) time-series techniques are given by Robert A. Stine, I.16, in *Time Series Models and Mathematica*. Once again, the integration of the techniques and graphics is very effective. We recall that time-series techniques also figured in Froeb's *Log Spectral Analysis* paper described above.

In *Data Analysis Using Mathematica*, Robert A. Stine, II.14, provides a rich set of data-analytical tools. Here are added to the *Mathematica* environment such graphical tools as histograms, kernel densities, box plots, QQ plots, and scatter plots, along with various smoothing and regression functions.

Finally, Colin Rose and Murray D. Smith, II.16, in *Random[Title]: Manipulating Probability Density Functions*, show how density functions can be transformed within *Mathematica*. Notions such as convolutions, marginalizations, conditionals, expectations and moments, moment-generating functions, and cumulants, both for single- and multi-variate random variables come alive in this work.

## Conclusions

It should be clear from the foregoing that, as an environment in which to carry out economic and econometric research, *Mathematica* is integrated, full-featured, powerful, versatile, extensible, and effective. Once one gains some control over the programming language, *Mathematica* provides an environment in which a great deal of one's research can be done. To my mind, it is one of a very small number of such packages that is worthy of a student's early and continued attention.

## Bibliography

- Amman, H. and D. Kendrick (1997), "Programming Languages in Economics", this volume, pp. xxx-xxx.
- Varian, Hal R. (ed.) (1993), *Economics and Financial Modeling with Mathematica*, Springer-Verlag (Telos): Santa Clara, CA.
- Varian, Hal R. (ed.) (1996), *Computational Economics and Finance*, Springer-Verlag (Telos): Santa Clara, CA.