May 3, 1995

Programming Languages in Economics[1]

by

David A. Kendrick and Hans M. Amman

Department of Economics
University of Texas
Austin, Texas 78712
USA
kendrick@mundo.eco.utexas.edu

and

Department of Macroeconomics
University of Amsterdam
Roeterstraat 11, Room 911
1018 WB Amsterdam
The Netherlands
amman@sara.nl

1. Introduction

Young economists sometimes ask which computer languages they should learn and more senior economists inquire about which language is most appropriate for a particular task. This paper attempts to provide some guidance by analyzing the comparative advantage of the programming languages which are most widely used by economists.

In the past, the phrase "programming languages" meant Fortran, Pascal and C. Today much of computer programming is done using high-level and modeling languages like GAUSS, Mathematica and GAMS, so a substantial part of this paper will be devoted to these languages. In addition, the paper will discuss a new class of languages for programming graphical user interfaces (GUI's) which are called "application frameworks". This class includes Visual Basic and Visual C++.

A key questions facing young economists today is how to navigate through the choices implicit in the paragraph above. Should one begin by learning Fortran and then C, or should one skip the low-level languages altogether and go straight to the high-level and

[1] We are indebted to Kit Baum, Jan Bisschop, Carl Chiarella, Arne Drud, Ken Judd, Jean-Pierre LeMaitre, Soren Nielsen, L. F. Pau, John Rust, Berc Rustem and Stavros Zenios for helpful suggestions.

modeling languages? Beyond this, one can ask whether the investment required to learn a programming language for a graphical user interface is worth the cost?

One guide in choosing programming languages to learn is the market test. It is useful to ask which languages are most used in a particular part of economics such as financial modeling, computable general equilibrium modeling or dynamic programming. Therefore, this paper includes references to the types of software used by various research groups.

Numerous questions have been asked in the paragraphs above. What are the answers? The paper is devoted to the answers but a brief summary of our advice is as follows:

> Begin with one of the high-level or modeling languages. For example, econometricians might begin by learning GAUSS, general equilibrium modelers, financial analyst and agricultural modelers might start with GAMS, and theorist with Maple, Matlab or Mathematica. Then work downward in the chain and learn either Fortran, Basic or C. Choose Fortran or C if you are interested in theoretical economics or in the solution of large and sparse models. Choose Basic, C or C++ if your goal is to move on to graphical user interfaces. Begin work on graphical interfaces with Visual Basic and then progress to Visual C++.

Obviously, there are many caveats to this advice. These caveats are supplied in the following which begins with the low-level languages and is followed by a discussion of a number of the high-level and modeling languages. The last section of the paper discusses the applications frameworks which are used by relatively few economists at present, but which could be very important in the future. The paper focuses on software which is used on personal computers since these are the machines which are most widely used by economists. While there are substantial groups of economists who use workstations, it seemed most useful to confine this paper to a discussion of PC software.

To facilitate an understanding of the various languages we will use small sections of code from a problem that we have coded in Fortran, GAUSS, GAMS and Visual C++. We do not use these pieces of code in this paper to compare solution speeds, since that is not our focus. Rather, we are interested here in comparing the *ease of learning* and *ease of use* of these languages.

The problem we use is the linear-quadratic control model which has linear systems equations and a quadratic criterion function. It is written as, find the control variables

$$\left(u_k\right)_{k=0}^{N-1}$$

to minimize the separation between actual and desired states and controls

$$J = \frac{1}{2}\left(x_N - \tilde{x}_N\right)' W_N \left(x_N - \tilde{x}_N\right) + \sum_{k=0}^{N-1}\left[\frac{1}{2}\left(x_k - \tilde{x}_k\right)' W_k \left(x_k - \tilde{x}_k\right) + \frac{1}{2}\left(u_k - \tilde{u}_k\right)' \Lambda_k \left(u_k - \tilde{u}_k\right)\right] \quad (1.1)$$

subject to the system equations

$$x_{k+1} = Ax_k + Bu_k + c \quad (1.2)$$

where

x_k = *state vector*
u_k = *control vector*
$\tilde{x}_k$ = *desired state vector*
$\tilde{u}_k$ = *desired control vector*
W_k = *state vector penalty matrix*
Λ_k = *control vector penalty matrix*
A, B, and c = parameter matrices and vectors

A part of the standard solution method using the first order conditions is the determination of the Riccati matrices K_k through the use of the equations

$$K_k = A'K_{k+1}A + W - \left[A'K_{k+1}B\right]\left[B'K_{k+1}B + \Lambda\right]^{-1}\left[B'K_{k+1}A\right] \quad (1.3)$$

which are integrated backward in time from the terminal condition

$$K_N = W_N \quad (1.4)$$

to the initial period. This is then coupled with a forward integration of the system equations (1.2) to provide the solution to the dynamic optimization problem.

2. Low-Level Languages

Over the years a substantial variety of languages have been used by economists, but Fortran has always persisted as the single most widely-used language. So the question has been whether to learn Fortran or one of the competing language of the time, whether it was Basic or Pascal or APL or C. Today the question is slightly more complicated. At the first level the question is whether to learn Fortran, Basic or C, but there is also the matter of whether to learn an AI language like LISP or Prolog and the question of whether or not to learn an object-oriented language like C++.

a. Fortran, Basic or C

Fortran is perhaps best know as the "language that does not die". Its demise has long been predicted, but it continues to be the weapon of choice for many economists. While inertia is one reason for its long life, a more important reason is the continuing technical progress in the development of Fortran compilers. Thus numerical methods programmed in Fortran continue to gain in efficiency not so much from clever new programming tricks as from the fact that improvements in the compilers make the language more efficient. One recent use of Fortran in this regard is by Rothwell and Rust (1995) who used Fortran to solve a dynamic programming problem because they felt that it was faster in loop operations than high level languages such as GAUSS or Matlab.

Also, the Fortran compilers have begun to support parallel processors and this is opening a whole new realm for efficiency gains. For example, Rothwell and Rust reported using Fortran for their dynamic programming problem in part because Fortran was the main language for obtaining the highest speeds and automatic vectorization on supercomputers such as the Cray C-90. For a survey of parallel processing in economics and with special attention to general equilibrium models and variational inequalities see Nagurney (1994). For applications of paralleling processing to stochastic control theory see Amman (1989), to the simulation of macroeconometric models see Gilli and Pauletto (1993), and to Benders decomposition for stochastic linear programming see Nielsen and Zenios (1994).

Also, a further gain in the efficiency of Fortran from a user's point of view is coming in the newest release of Fortran which is called Fortran 90. As an example, see Microsoft Fortran Powerstation NT for personal computers and workstations. Fortran 90 versions of the language supports matrix data types and dynamic memory allocation and therefore moves it into competition with some of the high-level languages like GAUSS.

One of the most important uses of Fortran in economics is to support numerical analysis methods like those described in Judd (1991). Not only is computational efficiency important in this area but also there are a large number of subroutine libraries such as IMSL, Netlib and NAG which can be readily accessed from Fortran code. Some of the work in this area can be described as "doing economic theory with numbers" so it is an area which had attracted economic theorists who have found limitations in analytical mathematics and who have extended their work by using numerical methods. This area includes not only numerical methods for solving linear and nonlinear equations but also approximation methods, numerical integration and dynamic programming.

Also, economists who are interested in optimization methods like linear, nonlinear and mixed integer programming as well as dynamic programming and control theory have been drawn to Fortran. Many of the best know optimization codes are programmed in Fortran. For example, a number of the linear, nonlinear and mixed integer programming codes which can be used with GAMS are programmed in Fortran, viz CONOPT, LSGRG2, and MINOS. Also, control theory codes such as our DUAL code are programmed in Fortran

as well as the code used by the group at Imperial College for solving nonlinear control models, c.f. Westcott, Zarrop, Holly, Rüstem and Becker (1979).

In order to make comparisons later in the paper with other languages, we provide here a few sections of code which would be used in solving the quadratic-linear control problem discussed above. The tradition in Fortran is to develop first a group of subroutines which do basic matrix operations and then to call these subroutines as needed to perform the operations. For example the subroutine below could be used to add the matrices A and B with the result being placed in the matrix C.

```
      subroutine madd (nr, nc, a, b, c)
      dimension a(nr,nc), b(nr,nc), c(nr,nc)
      do 105 j = 1,nc
      do 105 i = 1,nr
105   c(i,j) = a(i,j) + b(i,j)
      return
      end
```

where nr is the number of rows and nc is the number of columns in the matrices a, b and c. A similar subroutine call mmul provides matrix multiplication and one called mtran provide matrix transposition. $$ With the new Fortran 90 the above procedure can be simplified to

$$C = A * B$$

which is equivalent to what we see with higher-level languages. Then as a part of the matrix Riccati calculations in Eq. (1.3) it is necessary to calculate the matrix product

$$A'K_{j+1}A + W \qquad (2.1)$$

This is accomplished by the following subroutine calls

```
call mmul (n, n, n, kj1, a, ka)
call mtran (n, n, a, at)
call mmul (n, n, n, at, ka,atka)
call madd (n, n, atka, w, phi)
```

where n is the row and column dimension of the matrices, mmul multiplies the matrices kj1 and a to obtain the matrix ka, mtran transposes the matrix a to at, mmul multiplies the matrices at and ka to obtain the matrix atka and madd adds the matrices atka and w and stores the results in the matrix phi. From this small example one can see that solving the matrix Riccati Eq. (1.3) requires the development and debugging of a considerable amount of code.

Turning from the example and back to the general discussion of Fortran one can say that economists who are interested in computationally intensive methods for solving theoretical

problems involving only a few equations or applied optimization problems involving hundreds or thousands of equations will find Fortran their weapon of choice. Then what about Basic and C?

Basic used to viewed as the language which novices learned as a first computer language. While this may still be true, a new version of Basic has gained an innovative role in these days of graphical user interfaces. Visual Basic has recently taken an important role as the most accessible way to do Windows programming. With this language a newcomer can begin to put together a useful graphical user interface for his or her program in a relatively short period of time. Therefore economists who want to develop graphical user interfaces for their applications may be well-advised to begin by learning Basic and then working with Visual Basic.

Many feel that the C language is slowly, but surely, edging Fortran aside because of its widespread use in both PC's and workstations. C is also the language used to program most Windows applications. This tradition goes on now in the form of Windows NT and Windows 95 programming of "standard applications" for Windows. See for example Myers and Hamer (1993), Murray and Pappas (1993) or Richter (1995). These applications are written in C and are developed with the Visual C++ Workbench but do not make use of the Microsoft Foundation Classes nor of the AppWizard capability of Visual C++. However, they are fully capable of using components developed with C and C++. This "component ware" is rapidly gaining in importance.

b. LISP and Prolog

A few years ago there was a substantial interest in the application of "artificial intelligence" methods to problems in economics and finance, viz. Pau (1990) and Krishnan (1991). The languages of choice for these methods were LISP, cf. Winston and Horn (1984) and Prolog, cf. Clocksin and Mellish (1984). LISP had been used for many years but Prolog gained a substantial following only in the late 1980's. However, the interest in rule based AI methods has decreased, so these languages are not being used by many economists these days. However, an exception to this is Constraint Logic Programming (CLP) which could offer some nice opportunities in economics, see for example Pountain (1995).

It is also possible that these languages may make a come back among economists if the trend toward the use of software "wizards" continues. Examples of wizards who assist the user and in some cases learn the preferences of the user from observation are already occurring in major applications like the Excel spreadsheet. Also, it seems likely that wizard-like agents will operate over the Internet of the future to track down and retrieve information that is of interest to an individual. It seems logical that many of these wizards may be programmed in AI languages like Prolog and LISP.

c. C++

Object-oriented languages like C++ have come on fast in the last decade as the most important languages for the development of computer applications. In the past the central element of low-level languages was the subroutine which received data in arguments, transformed it and returned the results in other arguments. In contrasts, the central element of C++ is an object which is like a subroutine which contains data as well as code. Also, the objects belong to classes and support inheritance. For example, a programmer might define a class for arrays of numbers. Then an inherited class might be integer arrays and a further inheritance might be identity matrices.

An important example of the use of C++ in economics is the MatClass econometric system of Birchenhall (1994). One aspect of this language is that matrix objects are created and transformed to do both estimation and simulation of econometric models. An advantage of a low-level language like C++ relative to a high-level language like GAUSS for doing econometrics is that C++ is compiled while GAUSS is interpreted. This could lead to a substantial computational speed advantage for the C++ code. However, this advantage is offset somewhat by the fact that some of the numerically intense parts of GAUSS are coded in assembly language to take full advantage of the capabilities of floating point hardware. Also, one can purchase an add-on application module that creates an executable (non-interpreted) version of GAUSS.

Birchenhall's code may serve as a pathbreaker for economists in the sense that it opens the door for code written to model all kinds of economic entities as objects. The world of economics is a world of objects like commodities, plants, ports and banks. It is appealing to think of these groups in classes which inherit properties from one another. Therefore, it seems possible that some of the major economic applications of the future will be written in object-oriented languages such as C++. For example, one might imagine general equilibrium models populated by consumer and producer objects interacting to trade commodity objects.

Finally and perhaps most important C and C++ are increasingly being chosen as the languages in which to develop new applications. Also, C++ is crucial for learning to use one of the main "applications frameworks" which are discuss in the last part of this paper. However, before discussing these frameworks we turn to the high-level languages which are widely used in economics.

3. High-Level Languages

As was mentioned above the high-level languages are good place for the novice economists to start in learning programming languages. The reason is that much less effort is required in order to begin to be able to do useful work in these languages. For example GAUSS can be used for simple data transformation and estimation work within a short time after removing the shrink wrap. Similarly optimizing portfolio models or CGE

(computable general equilibrium) models can be solved with GAMS after relatively little training. Also, analytical derivatives of functions can be done with Mathematica or Maple soon after first using either of these products.

There are several reasons for the smaller start-up costs for using high-level languages. One is that much of the underlying complexity is hidden from the user. For example, one does not need to worry about memory management when using most of these products. Secondly, input and output is more simply done than in low-level languages. Thirdly, data types are at a higher level. For example GAUSS permits one to do operations directly on matrices rather than having to program do-loops to perform operations on the individual elements of the matrices. A fourth reason is the availability of examples which are close to the models used by economics researchers. This last reason deserves more discussion.

The best way to learn a high-level languages is to work through some sample calculations or models. For example, the GAMS system comes with a library of about one hundred models called GAMSLIB. These models range across many subject areas in economics from agriculture, to heavy industry, to computable general equilibrium models (CGE's), to finance and across levels of difficulty from simple to very complex models. Many new users of GAMS will find an example or two in their area of interest and can use these models as a quick way to get started with the language.

The same is true for most other modeling and high-level languages. Either the software will come with sets of sample models or the user's guides will contain the coding for a variety of uses of the software. This is a much easier way to begin than by reading the manual and in many cases there will be a sample which is close to the type of model the users wants to develop. Paul Samuelson told his students that economics was most efficiently done when it was possible to "stand on the shoulders" of one's predecessors. That is the case in spades with learning high-level programming languages.

In the discussion to follow high-level languages which are widely used in economics are classified into several areas, viz (1) econometrics, (2) optimization and CGE's, and (3) theory. While this partition is a useful starting point it must be used with care because most of the high-level languages can be used for a wide variety of purposes. For example GAUSS was originally developed as a language for estimation but it has substantial optimization capabilities beyond those needed strictly for estimation. Similarly GAMS was developed for optimization of large models but one can create in it almost any kind of model that can be stated algebraically. None-the-less, the partition used in this section serves as a good starting point for understanding the capabilities of the various high-level languages.

a. Estimation

While all econometrics packages include data manipulation and modeling languages of one form or another, the high-level language which has gained most widespread acceptance

recently is GAUSS (see Aptech (1993)). Therefore, we will not attempt to discuss the many econometric packages here, but rather confine the discussion to GAUSS and to the object-oriented econometrics package mentioned above, i.e. MatClass. For additional discussion of GAUSS and a comparison of GAUSS with MATLAB see Rust (1993).

i. GAUSS

GAUSS provides high-level input and output routines with real ease of use in interactive graphical analysis and plotting. For example, in the dynamic programming study by Rothwell and Rust (1995) much of the data preparation and analysis was done with GAUSS even though the dynamic programming problem was solved using Fortran. GAUSS has also been used by many econometricians for specifying estimation procedures, simulation methods and optimization algorithms.

One of GAUSS's most attractive features is the ability to express mathematical statements with matrices and vectors instead of do-loops. This gives GAUSS code a clean look which makes complex mathematical operations easy to read and check. An example is in Hatheway (1992) which contains elaborate matrix equations which are used to solve Riccati equations in a game setting concerning macro economic coordination between the U.S. and Japan. Another example is the nonlinear stochastic control code of Neck and Matulka (1994). Also, the quadratic-linear problem described above can be used to illustrate this point.

To input the terminal period state variable penalty matrix W_N in a GAUSS when this matrix is a 2 x 2 diagonal matrix with 100's on the diagonal it is only necessary to write

```
wn = {100   0,
        0 100};
```

It is not necessary to declare the matrix in advance nor give it dimensions. Rather the GAUSS program handles the declaration by inference from the input that wn is a matrix with 2 rows and 2 columns. Also GAUSS handles all the dynamic memory management for the user.

Then the terminal period Riccati equation matrix Eq. 1.4 can be written

```
kt1 = wn;
```

Once again it is not necessary to declare the matrix kt1 nor to dimension it. Rather GAUSS transfers the properties of the matrix wn to the matrix kt1. Then the Riccati Eq. 1.3

$$K_k = A'K_{k+1}A + W - \left[A'K_{k+1}B\right]\left[B'K_{k+1}B + \Lambda\right]^{-1}\left[B'K_{k+1}A\right] \qquad (3.1)$$

can be written in GAUSS with the statement

```
kt = a'*kt1*a + w - (a'*kt1*b) * invpd(b'*kt1*b + lambda) * (b'*kt1*a) ;
```

where

kt = K_k
kt1 = K_{k+1}
lambda = Λ
invpd = inversion operator

While the GAUSS statement above is not exactly the same as the mathematics in Eq. (3.1), it is close, and it is much simpler and more elegant than the Fortran code shown above. Of course the newest version of Fortran, i.e. Fortran 90, also has this capability. However, for many years this was a substantial advantage of GAUSS over Fortran. The progress of Fortran is this regard is great enough that some economists report porting code back from GAUSS into Fortran and find that the transition is rather minor.

GAUSS has also attracted many investigators to create add-on application modules. These include linear and nonlinear programming, game theory, rational expectations modeling, and control theory. With these add-on packages GAUSS now has powerful and easy to use capabilities in both estimation and optimization.

ii. MATLAB

In many respect Matlab is equivalent to Gauss. It is "algorithmically oriented" allowing you to make a function call that does a set of complicated operations by using one statement in your Matlab program. Variables are multi-dimensional objects that can be treated as scalars. Matlab runs fully under Windows and has a much nicer interface than Gauss with a large number of built-in graphical routines. Different than with Gauss, a Matlab program runs in a Windows box giving you the possibility to use the Windows GUI to display information and graphs. However, the set of numerical routines is somewhat limited in comparison to Gauss. So, depending on the type of problem, one may choose either Gauss or Matlab.

iii. MatClass

MatClass (Birchenhall (1994)) was described above as an application which was developed using the C++ language. It is discussed here as a high-level language which is a competitor to GAUSS. MatClass has classes not only for matrices and vectors but also for estimation procedures like ordinary least squares. A few lines of MatClass code can be used to illustrate something of its nature. Consider the code which is used to declare, input and multiply two matrices to obtain a third, i.e.

```
const INDEX m = 4, n = 3, p = 2 ;
matrix A(m,n), B(n,p), C ;

A.read() ;
B.read() ;
C = A * B ;
C.print() ;
```

The style with object oriented languages is to declare an object first as is done with the statement

```
matrix A(m,n), B(n,p), C ;
```

Each type of object has associated with it not only data but also methods which can be applied to the object. For example the statement

```
A.read() ;
```

uses the A matrix object and applies the read method, i.e. it reads the matrix from the data file.

Object oriented languages have operators whose meaning is related to the surrounding objects. For example in the statement

```
C = A * B ;
```

the * operator results in matrix multiplication. However, if A and B were scalars the * operator would result in scalar multiplication. Or if A was a scalar and B was a matrix the * operator would result in scalar time matrix multiplication. This is called an "overloaded" operator, i.e. an operator which can have different meanings depending on the context.

MatClass also has objects which are econometric estimators. For example an ordinary least squares estimation procedure done with single value decomposition is declared as

```
matrix x, y, beta ;
olsSvd ols ;
```

Data are then assigned and the coefficients of the regression computed with the statements

```
ols.assign ( y, x) ;
ols.coeff ( beta ) ;
```

In these two statements the first term is the object and the second term is the method to be applied to the object followed by the list of arguments. Thus the first statements indicates that the ols object should be used and that the assignment method should be applied to it by linking the matrices y and x to that estimation object. The second statement above applies the regression method to the ols object and places the coefficients in the beta matrix.

One of the unique aspects of MatClass is that Chris Birchenhall makes the source code for MatClass available to other economists and statisticians. This makes MatClass an invaluable tools on which other economists can build. Since C++ has an inheritance capability it seems likely that other economists and statisticians will add to the base classes for matrices and estimation procedures in MatClass in much the same way that the add-on application modules have been added to GAUSS.

b. Optimization and Computable General Equilibrium (CGE) Models

i. GAMS

Just as GAUSS begin as an econometric estimation package, GAMS (Brooke, Kendrick and Meeraus (1992)) began as a software package to solve algebraic models in general and optimization problems in particular. GAMS per se however, is not so much an optimization system as a modeling language which provides a powerful interface for a number of optimization codes. In the following section we will discuss GAMS as a modeling language. Here we will discuss it as a high-level language.

The principal creator of GAMS, Alexander Meeraus, called his software system the General Algebraic Modeling System, to emphasize the fact that almost anything you can do in algebra you can do in GAMS. Thus one can write in algebra the first order conditions for an optimization problem and use GAMS to solve this set of equations and thus the problem. Or a CGE model can be solved in GAMS by writing the first order conditions for consumer and producer optimization and coupling these with identities to provide a system on nonlinear equations which can be solved by GAMS.

However, while GAMS has been and will continue to be used in this way, it is also an important tools for economists as a modeling language and this is discussed below.

ii. MP Library

A C++ class library for mathematical programming which is called MP Library has been developed by Soren Nielsen (1994). The objects in this library are variables, constraints and models. For example the two variables and three constraints in a small model would be declared with the statements

```
MP_variable       x,  y ;
MP_constraint     time, bolts, screws ;
```

Then the constraints would be defined with the statements

```
time     =        .005 * x  + .007 * y  <=  40 ;
bolts    =                           x  <=  6000 ;
screws   =                           y  <=  4000 ;
```

In the above the words time, bolts and screws are not variables but rather the names of the inequalities. The equal sign following each of these words is not a part of the mathematics, but rather a symbol which divides the name of the constraint from its mathematical statement.

After the specification of the constraints the model would be declared and instantiated with the statements

```
MP_model  lathe ;
lathe.subject_to (time, bolts, screws) ;
lathe.objective = 25 * x  +  30 * y ;
```

Here the model object "lathe" is first operated on by the "subject_to" method to determine the constraints which belong to the model. Then it is operated on by the "objective" method to define the objective function.

Finally, the model is solved with the statement

```
lathe.maximize() ;
```

So the linear programming model "lathe" is maximized.

The underlying C++ code in MP Library includes efficient data structures for storing and manipulating sparse arrays. This is a capability which is extremely useful when one is solving large models where storing fully allocated arrays can be prohibitively expensive of computer memory.

In summary, Nielsen's MP Library offers a nice parallel for economists to Birchenhall's MatClass. The former is oriented toward mathematical programming and optimization and the latter towards econometrics. The availability of these two C++ systems opens the

door wide for economists who wish to do their programming in C++ but who do not want to write all the basic routines which are required to get started.

c. Theory

i. Macsyma

Many years ago computer scientists at M.I.T. developed a language called Macsyma which would take symbolic derivatives, perform integration and expand or reduce complex mathematical expressions. One of the early uses of this tool by economists was to compute symbolic matrix inversions for comparative statics analysis. For example, theoretical macromodels were specified as systems of three to six nonlinear equations and then linearized and solved symbolically in order to sign the effects of policy variables on target variables of interest. Macsysma was originally available on mainframes and then workstations.

ii. Mathematica

When personal computers came along the Macsysma tradition was carried forward by the Mathematica software. So a mathematical economists can begin his or her work with pencil and paper and then continue it with a personal computer to derive complex mathematical models or to do simple mathematical modeling experiments. For a discussion of some of the many uses of Mathematica by economists see Varian ((1993), (1995)). So the comparative advantage of Mathematica in economics is with small models of a few equations which begin as something to be solved with pencil and paper but which are too complicated to be reliably solved without the aid of computational tools.

However, Mathematica has also been used by some econometricians to good advantage. In this it becomes a competitor to GAUSS. For examples and discussion see Belsley (1993).

iii. Maple

Mathematica has been sufficiently successful that it has attracted a low-cost but high quality competitor in the form of Maple (1993). This language not only has most of the mathematical capabilities of Mathematica but also has its ability to provide output in the form of two and three dimensional plots.

In the following example we used Maple to analyze the properties of the cost-to-go function in an adaptive control model. We were concerned that the function might be nonconvex under certain conditions. The mathematics was so complex that we used Maple to check the algebra which had been done earlier by hand. Also, Maple provided an efficient system for doing three dimensional plots of the function.

At one stage in this work the cost-to-go function was in the form below (see Amman and Kendrick (1995))

$$J = (\psi_1 + \delta_1)u^2 + (\psi_2 + \delta_2)u + \psi_3 + \delta_3 + \frac{1}{2}\frac{\sigma^2 q \phi_1 (\phi_2 u + \phi_3)^2}{\sigma^2 u^2 + q} \qquad (3.1)$$

where

J = cost-to-go

u = control variable in time period zero

σ^2 = variance of a parameter which is being "learned"

q = variance of the additive noise in the system equation

$\psi's, \delta's,$ *and* $\phi's$ = parameters that are themselves functions of other parameters

Since the $\psi's$, $\delta's$ and $\phi's$ were themselves functions of the underlying parameters of the problem Eq. (3.1) was complicated when it was fully expanded and Maple provided a nice way to check the mathematics. Also, the three dimensional graphing capability of Maple proved to be most useful.

At one stage in the work we wanted to evaluate the second derivative of J with respect to the initial period control variable, u, at $u = 0$. This was done with the following Maple statements. The session is interactive with the input lines marked with the > prompt and the output lines indented and in mathematical form.

```
> J:= (f1+d1)*u^2 + (f2+d2)*u + f3 + d3 + (1/2)* s^2 * q * p1 * (p2*u+p3)^2/(s^2*u^2+q) ;
```

$$J := (f1 + d1)u^2 + (f2 + d2)u + f3 + d3 + \frac{1}{2}\frac{s^2 q\, p1 (p2u + p3)^2}{s^2 u^2 + q}$$

```
> Ju := diff (J, u) ;
```

$$Ju := 2(f1 + d1)u + (f2 + d2) + \frac{s^2 q\, p1 (p2u + p3)\, p2}{s^2 u^2 + q} - \frac{s^4 q\, p1 (p2u + p3)^2 u}{(s^2 u^2 + q)^2}$$

```
> Juu := diff (Ju, u) ;
```

(not shown)

```
> subs (u = 0, Juu) ;
```

$$2 f1 + 2 d1 + s^2 p1\, p2^2 - \frac{s^4 p1\, p3^2}{q}$$

Ju is the first derivative and Juu is the second derivative. We did not display the second derivative above since it is rather long, but we have shown the second derivative evaluated at u = 0. As one can see in the above, the recent addition of mathematical typefaces and

spacing to Maple have made the output much simpler to read than in the old system where powers and also the parts of fractions were displayed on separate lines.

iv. Combinations

Ken Judd has suggested that we should mention that there are some hybrid versions of the symbolic languages like Macsyma, Mathematica and Maple which will be on interest to some computational economists. For example, one can get a toolbox from Matlab which implements Maple. Also, there is a product which permits the combination of Matlab with Microsoft Word and another product which combines Maple with the Scientific Workplace word processor.

4. Modeling Languages

While most programming languages are used to provide a computer representation of an algorithm, modeling languages like GAMS are use to provide a computer representation of an economic model. For example, an optimization problem is usually solved with Fortran or GAUSS by using do-loops or matrix operations to code the algorithm which is used to solve the problem. The algorithm might be a variant of the simplex method for solving linear programs, a generalized reduced gradient method for solving nonlinear programming problems or a set of Riccati equations for solving quadratic-linear optimal control problems. However, in all these cases the programming language is used to code the steps and iterative procedures of the algorithm.

In contrast, the principal focus of a modeling language is on representing the economic model and not the algorithm for solving it. For example the focus in the GAMS representation of a model is on the sets of commodities, plants, markets and time periods and on the variables, equations and inequalities defined on these sets to represent the behavior of the economic actors.

In modeling language systems it is therefore possible to separate the model from the solver. For example, one can specify a nonlinear programming model in GAMS and then choose from a number of different "solvers" which embody different algorithms. Thus MINOS, CONOPT and GRG are different solver options that the GAMS user can activate to solve his or her model. Better yet, each of these solvers may be used in turn to determine which is best at solving the model under investigation.

This separation also has the great value that the economist who wants to solve optimization models need not become an algorithm expert and can rather focus his or her attention on the modeling of economic behavior. Moreover, as technical progress occurs in algorithms and new and improved solvers are embedded in the modeling language system, the model does not have to be altered, rather only the statement selecting the solver.

While rudimentary modeling languages date from the TROLL system and the early versions of TSP and other econometric packages, it is in the optimization and CGE systems like GAMS and GEMPACK that these languages have come into their own.

a. GAMS

GAMS is set driven. This is in contrasts to GAUSS which is matrix driven. In GAMS one defines sets of commodities, time periods, plant, markets and commodities and then specifies the variables and equations of the model over these sets. Thus it is not necessary to handcraft each equation in the model, but rather only to enter types of equations defined over the sets. Thus in a CGE model there would be a market clearing equation defined over commodities and time periods. The model might have twenty commodities and ten time periods, but the GAMS input would have only a single equation defined over the sets of time periods and commodities. This single equation would then be expanded to the 200 equations of the model.

The set driven nature of GAMS makes it an intuitive and powerful modeling language for large scale economic models such as agricultural or industrial models, financial models and CGE's. For a discussion of some of the financial models in GAMS see Dahl, Meeraus and Zenios (1993), for CGE's see Rutherford (1992) and for sectoral models and many others see the GAMS library which is distributed with the software.

However, GAMS is more awkward to use when the model can be expressed purely as a set of mathematical operations on matrices. So the strong point of GAUSS is the weak point of GAMS. In fact it is useful to think of GAUSS as an algorithm language and of GAMS as a modeling language. The algorithms in GAMS are optimization packages which are provided along with the basic system. For example MINOS, GRG, CONOPT, ZOOM and MPSX are a few of the optimization packages which can be integrated into GAMS. These codes range across linear and nonlinear programming to mixed-integer and nonlinear mixed integer programming procedures.

Now consider the specification of the quadratic-linear problem in the GAMS language. For this example, we use a simplified version of some section of Chartchai Parasuk's optimal control model of the U.S. economy in Parasuk (1989). Begin with the sets of state variables, control variables and time periods.

```
sets    n       states          /       c       consumption
                                        i       investment
                                        yd      disposable income /

        m       controls        /       g       government expenditures
                                        t       tax rates
                                        omo     open market transactions /
```

```
k       horizon       /       81-1,  81-2,  81-3,  81-4 /
```

Here we have used three state variables, three control variables and four time periods. In Parasuk's original model there are fourteen states, six controls and seventeen time periods. While the small example above is sufficient to illustrate model specification in the GAMS language, Parasuk's larger example would have been useful to illustrate the power of set-driven languages like GAMS in specifying large models.

GAMS also has the ability to make copies of sets which are useful when doing making operations. For example the statement

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

serves to create a copy np (n prime) of the set of states and a copy mp (m prime) of the controls.

The variables of the model are then defined over the sets with the statements

```
variables     x (n, k)        state variables
              u (m, k)        control variables
              J               criterion value
```

With this notation in hand one can write a portion of the quadratic criterion of Eq. (1.1),

$$J = \frac{1}{2}\sum_{k=0}^{N}\left[\left(x_k - \tilde{x}_k\right)' W_k \left(x_k - \tilde{x}_k\right)\right] \qquad (4.1)$$

in GAMS as

```
criterion..     J =e= .5 * sum( (k, n, np),

                    ( x(n,k) - xtilde(n,k) ) * w(n, np, k) * ( x(np,k) - xtilde(np,k) ) ) ;
```

where criterion is the name of the objective function and the sum in the right hand side of the equation is over all three sets, i.e. the set of time periods, k, the set of states, n, and the copied set of states, np.

Then after the systems equations are entered one writes

```
model  exchange / all / ;
```

and

```
solve exchange minimizing J using nlp ;
```

The first of these two statements assigns the name exchange to the model which consists of all of the equations above. The second statement instructs GAMS to solve this model by minimizing the criterion value J using a nonlinear programming algorithm.

Here the focus is not on the algorithm to solve the model, but rather on the representation of the model itself. The default nonlinear programming solver in GAMS is MINOS by Murtagh and Saunders (1987) so the solve statement will apply that code and its embodied algorithm to the model. If one preferred instead to solve the model with Lasdon's (1978) GRG code or Drud's (1985, 1994) CONOPT code one would insert one of the following two lines just above the solve statement

```
option nlp = grg ;
```

or

```
option nlp = conopt ;
```

This separation of the model representation from the solvers is an important change in programming languages. The same kind of division can be accomplished with GAUSS by using some of the add-on modeling components. And, by the same turn, GAMS can be used to program an algorithm. For example, the Riccati equation approach to solving quadratic linear problems has been implemented in GAMS by Soren Nielsen.

GAMS has been widely used by for CGE models. An elementary treatment of the use of GAMS for CGE models is in Kendrick (1990). Recently Rutherford (1992) has provided a link between GAMS and MPSGE to create a powerful CGE modeling system which can handle not only equality but also inequality relationships.

There is a covey of modeling languages which are competitors to GAMS. The oldest are AMPL by Fourer, Gay and Kernighan (1992) and Structured Modeling by Geoffrion (1992) and the most recent is AIMMS by Bisschop and Entriken (1993). The AIMMS system is distinguished by providing a graphical end-user interface builder which makes it easy to extend the use of models to end-users who are familiar with the application but do not have to know about the inner workings of the model. AIMMS also has a number of new features for the management of data, reports, graphics and views. In addition AIMMS has a GAMS-compatible mode thereby extending the use of existing GAMS models to the domain of end-user applications.

b. GEMPACK

GEMPACK (1993) was originally created by Ken Pearson in Fortran as the modeling system and solver for the Johansen style CGE models which were developed by the Peter Dixon and Alan Powell group at Melbourne and Monash Universities in Australia, see Codsi and Pearson (1988). The early versions of this software were able to solve large CGE's containing thousands of equations very efficiently, but were restricted to static models. Also, the models had to be specified in a linearized form in which the variables were in percentage-rate-of-growth form in the Johansen tradition. However, in recent years the software has been modified to include models in which the variables can be specified either in percentage rate of change, in the ordinary levels manner with linear or

nonlinear equations, or in a mixture of the two modes, cf. Harrison, Pearson, Powell and Small (1994). Also, the software can now be used to solve dynamic CGE models.

Like GAMS, GEMPACK also makes use of sets. For example the statements

```
SET REG  # regions # (usa, row) ;

SET COMM  # commodities  # (food, manuf, services) ;

SET TRAD_COMM  # traded commodities # (food, manuf) ;
```

would be used to create the sets for a model with two regions, three commodities and two traded goods. Then a group of activity level variables would be created with the statement

```
VARIABLE (all, i, COMM) (all, r, REG)   pm(i,r) ;

VARIABLE (all, i, TRAD_COMM)   pw(i) ;

VARIABLE (all, i, TRAD_COMM) (all, r, REG)   tt(i,r) ;
```

where pm is the market price, pw is the world price, and tt is the tariff. Then an equation can be specified in GEMPACK in the form

```
EQUATION MKTPRICES

(all, i, TRAD_COMM) (all, r, REG)

pm(i,r) = tt(i,r) + pw(i) ;
```

The variables here are in percentage rates of change so this equation says that the percent change of the market price will be equal to percent change in the tariff plus the percent change in the world price. The statements above are modifications of an example model in Hertel, Horridge and Pearson (1994).

Economists who are interested in developing large scale CGE's find GEMPACK a powerful aid to their work not only for the software itself but also for the tie to the long running IMPACT project which has developed a series of models of the Australian economy and also participated and encouraged the development of both environmental and trade-oriented CGE's in the U.S. and South Africa as well as Russia, cf. Dixon, Parmenter, Powell and Wilcoxen (1992).

5. Application Frameworks

In the early 1960's an economists could prepare a computer program for use by himself or herself and colleagues and students by writing and debugging a main routine and a set of

subroutines. Some of these routines handled input and output and were something of a pain to get right because of the exacting demands of "format" statements; however, the task was straightforward, if tedious. The resulting program would accomplish the task but was usually difficult for others to use because the documentation was often spotty and even when the documentation was well done, a considerable effort was required in learning to use the program.

Today, one can write applications that are much easier to use and where the entry cost is very low. While documentation is still required, the graphical user interfaces available now make it possible for the user to experiment and point and click in most programs without spending much time with the documentation. However, the process of programming the user interface can either be fairly simple or extremely complicated and time consuming. This section describes the "application frameworks" which are used to prepare graphical user interfaces for applications and provides some guidance about the level of difficulty involved in various approaches.

The two approaches which are widely used on personal computers are Visual Basic and Visual C++. The first is considerably easier to learn and easier to use for implementing simple applications. The second has a much steeper learning curve, but leaves one free to develop applications of any degree of complexity.

a. Visual Basic

As was discussed above, the Basic language was originally viewed as a simple language which was good for the beginner to learn and then to set aside as he or she moved on to Pascal or Fortran. It was also expected that Basic would not have a long lifetime. However, like Fortran, Basic did not die. Rather with a substantial input of resources from the Microsoft Corporation and the personal care and attention of Bill Gates and many others, the Basic language grew, developed and flourished.

Recently, Microsoft has created Visual Basic which can be used to create a graphical user interface for applications (see for example Craig (1994)). One only has to learn the fundamentals of the Basic language and the procedures for creating windows and dialog boxes and a workable graphical interface can be created. Moreover, even the design of the dialog boxes and of aspects of the windows such as the menus can be done graphically and this greatly speeds the work of developing the interface.

Not only are windows and dialog boxes easy to create, but also a small industry has grown up to provide "components" for Visual Basic. These components are called Visual Basic controls (VBX). They enable the programmer to gain easy access to a variety of controls which do important parts of the work of the graphical interface. Examples are providing a simple spread-sheet to manage input of data or controls for multimedia capabilities like video and audio.

So the programmer armed with the Visual Basic software and access to VBX components may be able to put together a simple interface for his or her application in a relatively short period of time. Many economists will find these interfaces sufficient. Others will want more substantially capabilities and will turn to Visual C++.

b. Visual C++

At one level Visual C++ can be viewed as a simple "programmer's workbench" which consists of an integrated editor, compiler, linker and browser supporting the use of C language programs and calling Windows routines to develop "standard applications". At another level it is a sophisticated development system for object-oriented C++ code relying on class libraries and wizards to speed the development of applications which can include object linking and embedding (OLE).

i. Standard Applications

For someone who already has some knowledge of the C language the most straightforward use of Visual C++ is to use the editor, compiler and linker of that system to write code which calls the windows subroutines in the software development kit (SDK) *without* using C++ or the Microsoft Foundation Classes (MFC). While C is simpler than C++, learning to develop standard applications in the C language is not easy since there are hundreds of subroutines in the Windows API (Application Program Interface). However, there is an excellent book by Petzold (1992) which provides an introduction to this subject along with a well integrated set of sample programs. Petzold's book was written for 16-bit systems, but Microsoft has supplied manuals for the 32-bit system to ease the transition, Microsoft Corporation (1993). Also, there are a number of other helpful books on Windows NT and Windows 95 programming including those of Myers and Hamer (1993), Murray and Pappas (1993) and Richter (1995). One advantage of this system is that one can target not only Intel platforms but also Alpha and PowerPC platforms.

As an example of the use of a standard application consider a user interface called DUALI which we have developed for both deterministic quadratic-linear models and dual control stochastic models. Though this work is at an early stage it can be used to illustrate some of the potential of graphical user interfaces (GUI). In the Fortran version of DUAL the second line of the input data looks something like

```
2   2
```

These two numbers are the number of state and control variables in the model. Then a little further down in the input is

```
.033  -.002
.124  -.045
```

which is the input for the B matrix in the quadratic-linear control model. While this is a perfectly satisfactory form of input for the seasoned user and for small models, it is not so intuitive as the input forms which can be provides with graphical user interfaces. Also, it is more likely to result in erroneous input than data supplied to a GUI.

In contrast, in DUALI one selects a menu to set the size of the model and a Size Dialog Box appears with boxes for inputting the dimensions.

Size Dialog Box

states	2	controls	2

While it would be difficult to get confused about the size elements in the deterministic quadratic linear control model, in the dual control models the users must enter a considerable number of elements in this dialog box and the GUI can be a substantial help in providing the context for each entry and thereby decreasing the learning term and the error rate.

Next the user opens a dialog box to enter the acronyms and names for the variables as follows:

Acronyms and Names

States

cons	consumption expenditures
inv	investment

Controls

money	money supply
gov	government expenditures

Following this the user opens a dialog box to enter first the A matrix and then the B matrix for the quadratic linear model. The acronyms are supplied in the spreadsheet-like input and this serves to decrease errors in coefficient inputs. So the B matrix look like

	money	gov
cons		
inv		

While these row and column labels are unimportant in a models with two states and controls, in larger models they provide an important additional safeguard. After the user has entered the data the result is as follows:

	money	gov
cons	.033	-.002
inv	.124	-.045

Each of the boxes here is itself a small edit window and can be equipped with copy, cut and paste functions to facilitate rapid data entry. The entire set of small windows can be scrolled to facilitate the entry of large data matrices.

The GUI described above is a small beginning of what is to come in this field. For example, we have begun experimentation with parallel model representations, viz. Kendrick (1995). Under this paradigm the user has a number of different representation of the model available in various windows open on the computer display. An example is shown in Fig. 5.1.

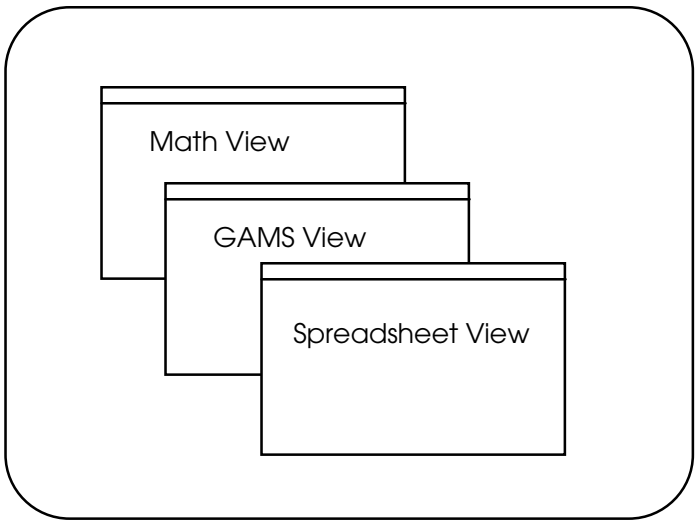

Figure 5.1 Parallel Model Representations

One of the windows might contain a mathematical description of the model like Eqs (1.1) - (1.2). Another would be a modeling language representation like GAMS or a high-level language representation like GAUSS or MatClass. Another would be in a spreadsheet form like the matrix data entry discussed above. Another would use a flow chart diagram of the sort commonly used by aeronautical and electrical engineers. Another might use a database approach coupled with an artificial intelligence capability.

These views would be updated simultaneously, so that when the user changes the functional form in the mathematical view this ripples through the other windows showing,

for example, the new functional form in the GAMS window. These parallel representation windows then permit the user to work on various aspects of the model in the representation which is most natural for that attribute. For example, the user might modify data tables using the spreadsheet view, functional forms using the math view and data transformation using the GAMS view.

With this kind of an input environment and a library of models one can bring a novice user up to speed in first a deterministic and then a stochastic dual control algorithm in much less time than is required with the traditional form of input. So at the first level GUI's are an important way to decrease entry cost for the novice. At the second level there are most useful to the expert in decreasing the probability of errors in the input and the use of the software.

When you also provide a graphical capability to display the results of the model calculations, one has a system in which the user can quickly and easily make changes to the parameters, re-solve the model and compare the effects of the parameter change on the results of the model calculations.

So much for the example - now we return to the general discussion of standard applications. The advantage of beginning with standard applications in programming graphical user interfaces (GUI's) is that this approach permits one to learn the use of the Windows subroutines first and separately without having to master object-oriented programming and the Microsoft Foundation Classes at the same time.

Developing standard applications therefore offers an intermediate approach between Visual Basic and the full Visual C++ with class libraries and wizards. Also, it provides an avenue for the use of components such as the new controls with are now being developed under Visual C++. Thus full access to the capabilities of 32-bit Windows software is open to the programmer without having to make a large investment in object-oriented programming and class libraries.

This approach also provides a smooth path to the full Visual C++ framework since the compiler permits one to mix C and C++ statements and this opens the door to the use of the class libraries.

ii. MFC/ Wizard/OLE Applications

If one has little previous experience with the C language or with Windows programming the best way to begin might be to take the plunge directly into Visual C++ using the wizards of that system and classes from the Microsoft Foundation Classes library. This is a little like learning to ski by going first to the black slopes on the highest hill, however, it might be the shortest path to developing industrial strength graphical interfaces for economic applications.

It is also a beguiling approach. One can begin with little or no knowledge of C or C++ or Windows programming and follow the step-by-step instructions and examples in a book like Perry, et al (1994) and soon be developing nice interfaces. So long as the interface requirements are fairly close to those outlined in the book one can proceed without too much difficulty. However, when one ventures away from the well-trodden path outlined by the book, serious difficulties may arise. When this occurs the programmer discovers that he or she needs a sound understanding of the following subjects:

1. Programming with the C language
2. Object-oriented programming with the C++ language
3. Windows NT
4. Windows programming
5. The Microsoft Foundations Class Library
6. The Visual C++ System with Wizards

Therefore, the risk for the novice in using the Visual C++ system is that tremendous complexity underlies the smoothly working examples in the Visual C++ books. One can develop fairly elaborate graphical interfaces without much understanding of all the underlying complexity so long as the methods used follow closely those outlined in the books. However, when one strays from the well-documented path, suddenly it becomes necessary to have a substantial understanding of most of the subjects mentioned above.

The six items in the list above include Windows NT because the most recent version of Visual C++, i.e. Version 2.0 requires Windows NT Version 3.5. So the programmer may need to investment first in learning Windows NT before installing Visual C++ and the Software Development Kit.

For an example of the use of Visual C++ to develop an interface for economics software see Chiarella and Khomin (1995). Their system which is called SND (Stochastic Nonlinear Dynamic) TOOL is a program which offers a user friendly environment for simulating systems of stochastic ordinary differential equations. At present their system is specialized for the use of a nonlinear stochastic model of exchange rate dynamics. They use a discrete time version of their model and provided two different versions of the program, depending on which parameter agents are trying to learn: the differential interest rate or the speed of adjustment. The software is designed in such a way that it will probably be modified to handle more general applications in the future.

So what is our advice to the novice about using applications frameworks? Begin by learning a little of the Basic language and use this to boost yourself into using Visual Basic. Develop a few simple interfaces with this system and if they are adequate for your needs then stop. If they are not adequate then consider two paths.

The first path is to learn the C language. Then proceed to Windows programming by developing a “standard application”. Next learn the C++ language and apply this

knowledge to master the Microsoft Foundation Classes. Then you can proceed with the Application Wizard and Class Wizard in Visual C++ to develop applications. The second path by-passes the C language and goes directly to C++. Then with some knowledge of C++ one is ready to use the Microsoft Foundation Classes in Visual C++ along with the wizards to develop applications.

Having mastered all of this what is next?

iii. OLE Applications

Windows programming can be very time consuming even with all the help that is provided by the class libraries and the wizards. One reason for this is that many applications require the development of sections which are like database systems or spreadsheets or word processors and each of these sections can be a substantial programming project by itself. The way around this is to use object-linking and embedding - OLE - pronounced like the Spanish ole'. This capability which is available in Version 1.5 and Version 2.0 of Visual C++ permits one to insert spreadsheets, databases or word processing sections directly into the documents created by the user's application and to use all the code from these applications in preparing the objects. Figure 5.2 shows an example in which there is text processing document which also contains a spreadsheet and a figure.

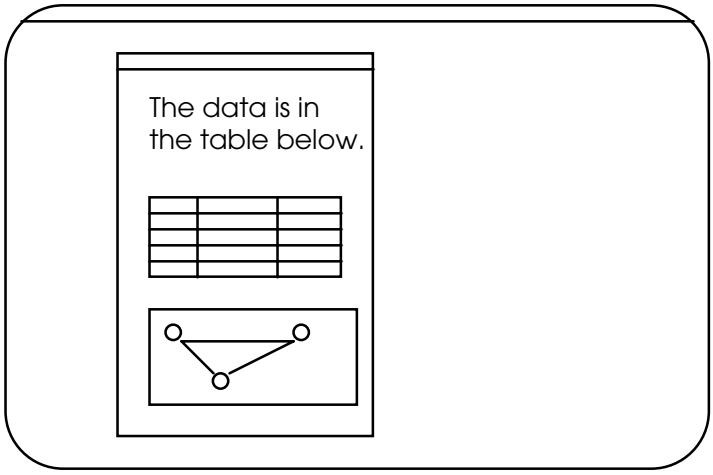

Figure 5.2 An OLE Example

When the user selects the spreadsheet or the figure, the menu bar is taken over by that application and the user can modify that component just as though he or she were working in that application.

This is a very attractive idea and one that could have profound and lasting affect on software development. However, the jury is still out. At first it was difficult to develop OLE applications but this is rapidly becoming easier. The concern remains however that if an economists develops software which ties an application to programs created by

others, then the application will require high-levels of maintenance to keep it running as the applications to which it is linked continue to change. So it could be that OLE will decrease the fixed cost of developing applications but greatly increase the maintenance costs to keep them running.

5. Conclusions

We end where we started. The best advice for the novice is to learn first a high-level language or modeling language which focuses on his or her special areas of interest in economics. Then one should learn a low-level language. If one wants to develop quick and easy interfaces for the Visual Basic application framework then the choice should be Basic, but with the expectation that you may also want to learn Fortran or C at a later date. If one expects to be modifying existing code, then the choice will most likely be Fortran since so much economic software has been written in that language. If one expects to start anew, then the choice will most likely be C because it matches Fortran in portability, efficiency and flexibility and opens the highway both to the application frameworks and to the C++ language.

Software developed without good graphical user interfaces is not likely to be used widely, so if one aims for broader usage, then learning one of the applications frameworks like Visual Basic or Visual C++ is well advised.

We should mention one other caveat about choosing which programming language to learn. Jean-Pierre LeMaitre has old us that it is his experience that begining programmers learn most quickly when they choose a language that it is used by others in their immediate environment. Having many other people available to help with the range of questions which ensue is extremely valuable.

So what does the future offer? Progress in high-level and modeling languages is making it increasingly easy for economists to do their own programming. In the past many economists hired an assistant to do most of their programming and were in great trouble when that assistant moved on to other pursuits. Now it is possible for economists to get up to speed in a high-level and modeling language like GAMS, GAUSS or Mathematica rather easily and to do their own programming. In fact many economists learn several of these high level languages and find that the combination provides them with great facility to program a wide variety of types of economic models.

At the same time the arrival of graphical user interfaces is offering important new opportunities to make economics software even easier to use than the high-level languages. These interfaces can be developed in simple cases with relatively little learning time by using Visual Basic or even Visual C++. In more complicated cases it is important for the economists to make a substantial investment in programming languages in order to learn the fundamentals which underlie these graphical interface development systems. However, the payoff can be substantial. One can imagine that the economics software of

the future will extend graphics to annimation and the computational environment of the economists of the future will be very different than the one we know today. A knowledge of high-level languages is likely to the key to developing software for these environments.

References

Amman, Hans M. (1989), "Nonlinear Control Simulation on a Vector Machine," *Parallel Computing*, Vol. 10, pp. 123-127.

Amman, Hans M. and David A. Kendrick (1995), "Nonconvexities in Stochastic Control Models," forthcoming in the *International Economic Review*.

Amman, Hans M., David A. Kendrick and John Rust (1995*), Handbook of Computational Economics*, forthcoming North-Holland Publishing Company, Amsterdam.

Aptech Systems Inc. (1993), GAUSS, Maple Valley, Washington, 98038.

Belsley, David A. (1993), "Econometrics.m: A Package for Doing Econometrics in *Mathematica*," in Varian (1993).

Belsley, David A. (ed) (1994*), Computational Techniques for Econometric and Economic Analysis*, Kluwer Academic Publishers, Dordrecht, The Netherlands.

Birchenhall, C. R. (1994), "MatClass: A Matrix Class for C++", in Belsley (1994), pp. 151-172.

Bisschop, Johannes and Robert Entriken (1993), *AIMMS: The Modeling System*, Paragon Decision Technology, 2001 DG Haarlem, The Netherlands.

Brooke, Anthony, David A. Kendrick and Alexander Meeraus (1992*), GAMS, A Users Guide, Release 2.25*, The Scientific Press Series, Boyd and Fraser Publishing Company, Danvers, MA 01923.

Chiarella, Carl and Alexander Khomin (1994) "A Nonlinear Stochastic Model of Exchange Rate Dynamics", Working Paper, School of Finance and Economics, University of Technology, Sydney, PO Box 123, Broadway 2007, NSW, Australia.

Clocksin, W. F. and C. S. Mellish (1984), *Programming in Prolog*, Second Edition, Springer-Verlag, Berlin.

Codsi, G. and K. R. Pearson (1988), "GEMPACK, General-Purpose Software for Applied General Equilibrium and Other Economic Modellers," *Computer Science in Economics and Management*, Vol. 1, pp. 189-207.

Cooper, William W. and Andrew B. Whinston eds. (1994*), New Directions in Computational Economics*, Kluwer Academic Publishers, Dordrecht, The Netherlands.

Craig, John Clark (1994), *Microsoft Visual Basic Workshop: Windows Edition*, Microsoft Press, Redmond, Washington 98052-6399.

Dahl, H., A. Meeraus, and S. A. Zenios (1993), "Some Financial Optimization Models: Risk Management," pp. 3-36 in S. A. Zenios (1993), *Financial Optimization*, Cambridge University Press, Cambridge, England.

Dixon, P. B., B. R. Parmenter, A. A. Powell and P. J. Wilcoxen (1992*), Notes and Problems in Applied General Equilibrium Economics*, North-Holland, Amsterdam.

Drud, Arne (1985), "CONOPT - A GRG Code for Large Sparse Dynamic Nonlinear Optimization Problems," *Mathematical Programming*, Vol. 31, p. 153.

Drud, Arne (1994), "CONOPT - A Large-Scale GRG Code*", ORSA Journal on Computing*, Vol. 6, No. 2, pp. 207-218.

Fourer, R., D. M. Gay and B. W. Kernighan (1992*), AMPL: A Modeling Language for Mathematical Programming*, The Scientific Press Series, Boyd & Fraser Publishing Co., Danvers, MA. 01923.

GEMPACK (1993), GEMPACK Manager, Impact Project, Menzies Building, Monash University, Clayton, Victoria 3168, Australia.

Geoffrion, A. M. (1992), "The SML Language for Structure Modeling: Levels 1 and 2," *Operations Research*, Vol 40, pp. 38-57.

Gilli, M. and G. Pauletto (1993), "Econometric Model Simulation on Parallel Computers," *The International Journal of Supercomputer Applications*, Vol. 7, No. 3, pp. 254-264.

Hatheway, Lawrence Howell (1992), *Modeling International Economic Interdependence*, Ph.D. Dissertation, Department of Economics, Univ. of Texas, Austin, Texas 78712.

Harrison, W. Jill, K. R. Pearson, Alan A. Powell and E. John Small (1994), "Solving Applied General Equilibrium Models Represented as a Mixture of Linearized and Levels Equations," *Computational Economics*, Vol. 7, No. 3, pp. 203-223.

Hertel, Thomas W., J. Mark Horridge and K. R. Pearson (1994), "Mending the Family Tree: A Reconciliation of the Linearization and Levels Schools of AGE Modelling," *Economic Modelling*, Vol. 9, pp. 385-407.

Holly, Sean, Berç Rüstem and Martin Zarrop (1979), *Optimal Control for Econometric Models*, The MacMillan Press Ltd, London.

Judd, Kenneth L. (1991), *Numerical Methods in Economics*, draft book from Hoover Institution, Stanford University, Stanford, California 94305.

Kendrick, David A. (1990), *Models for Analyzing Comparative Advantage*, Kluwer Academic Publishers, Dordrecht, The Netherlands.

Kendrick, David A. (1995), "Ten Wishes," *Computational Economics*, Vol. 8, No. 1, February, pp. 67-80.

Krishnan, Ramayya (1991), "PDM: A Knowledge-based Tool for Model Construction," *Decision Support Systems*, Vol. 7, pp. 301-304.

Lasdon, Leon S., Alan D. Warren, A. Jain and M. Ratner (1978), "Designing and Testing of a Generalized Reduced Gradient Code for Nonlinear Programming," *ACM Transactions on Mathematical Software*, Vol. 4, p. 34.

Maple V, Release 2 (1993), Brooks/Cole Publishing Co., Pacific Grove, California, 93950.

Microsoft Corporation (1993), *Microsoft Win32 Programmer's Reference*, Microsoft Press, Redmond, Washington 98052-6399.

Murray, William H. and Chris H. Pappas (1993), *Application Programming for Windows NT*, Osborne McGraw-Hill, New York.

Murtagh, Bruce A. and Michael A. Saunders (1987), "MINOS 5.1 User's Guide," Report SOL 83-20R, December 1983, Revised January 1987, Stanford University, Stanford, CA.

Myers, Brian and Eric Hamer (1993), *Mastering Windows NT Programming*, Sybex, San Francisco.

Nagurney, Anna (1994), "Parallel Computation", chapter in Amman, Kendrick and Rust (1995).

Neck, Reinhard and Josef Matulka (1994), "Stochastic of Nonlinear Economic Models," in Cooper and Whinston (1994), pp. 207-226

Nielsen, Soren S. (1994), "A C++ Class Library for Mathematical Programming," forthcoming in A. Sofer and S. G. Nash (eds*), The Impact of Emerging Technologies on Computer Science and Operations Research*, Kluwer Academic Publishers, Dordrecht, The Netherlands.

Nielsen, Soren S. and Stavros A Zenios (1994), "Scalable Parallel Benders Decomposition for Stochastic Linear Programming," working paper, Management Science and Information Systems, University of Texas, Austin, Texas 78712.

Parasuk, Chartchai (1989), *Applications of Optimal Control Techniques in Calculating Equilibrium Exchange Rates*, Ph.D. dissertation, Department of Economics, Univ. of Texas, Austin, TX 78712.

Pau, L. F. (1990), "A Survey of Reasoning Procedures in Knowledge Based Systems for Economics and Management," *Computer Science in Economics and Mgmt*, Vol. 3,p.3-22.

Perry, Paul J., Chris Corry, Chane Cullens, Mark Davidson, Robin W. McKean and John Tackett, Jr. (1994), *Using Visual C++ 2*, Que Corporation, Indianapolis, IN 46290.

Petzold, Charles (1992), *Programming Windows 3.1*, Third Edition, Microsoft Press, Redmond, Washington 98052-6399.

Pountain, Dick (1995), "Constraint Logic Programming," *Byte*, February, Vol. 20, No. 2, pp. 159-160.

Richter, Jeffrey (1995), Advanced Windows: *The Developer's Guide to the Win32 API for Windows NT 3.5 and Windows 95*, Microsoft Press, Redmond, WA 98052-6399.

Rothwell, Geoffrey and John Rust (1995), "A Dynamic Programming Model of U.S. Nuclear Power Plant Operations," mimeo, Department of Economics, Univ. of Wisconsin, Madison, WI.

Rust, John (1993), "GAUSS and MATLAB: A Comparison", *Journal of Applied Econometrics*, Vol. 8, pp. 307-324.

Rutherford, Thomas F. (1992), "Extension of GAMS for Complimentarity Problems Arising in Applied Economic Analysis," working paper, Department of Economics, Univ. of Colorado, Boulder, Colorado 80309.

Stahl, Dale O. and Andrew B. Whinston (1994), "A General Equilibrium Model of Distributed Computing," in Cooper and Whinston (1994).

Varian, Hal (1993), ed. *Economics and Financial Modeling with Mathematica*, TELOS/ Springer-Verlag, New York.

Varian, Hal (1995), "Mathematica for Economist," chapter in Amman, Kendrick and Rust (1995).

Westcott, John, M. B. Zarrop, S. Holly, B. Rüstem and R. Becker (1979), "A Control Theory Framework for Policy Analysis," Chapter 1 in Holly, Rüstem and Zarrop (1979).

Winston, Patrick Henry and Berthold Klaus Paul Horn (1984), *LISP*, Second Edition, Addison-Wesley, Reading, Massachusetts.