Object-Oriented Econometrics: Matrix Programming in C++ using GCC and Newmat

Dirk Eddelbüttel, G.R.E.Q.A.M., Marseilles, France

Summary

Object-oriented programming using C++ offers features that can be as beneficial for econometricians as they are for other programmers. This review considers the standard programming paradigm and then discusses object-oriented programming, in particular the C++ language. Both GCC, a C++ compiler available free of charge from the GNU Project, and the newmat matrix class library are reviewed. An example of a simple econometric program in C++ that uses newmat is included.

1 Introduction and Overview

Econometricians frequently write computer programs. This article reviews the possibilities offered by a C++ compiler together with an appropriate matrix class. The C++ compiler that we review here is GCC and the matrix class that we have chosen is newmat. Both are available at no charge on the Internet.

Over the years, an extensive literature has developed on approaches to programming in general, on object-oriented programming in particular, and on the C++ language. Within this review, however, space constraints prevent us from covering more than only the surface of the most important aspects of object-oriented and C++ programming, but we provide pointers for further information.

In order actually to *use* a programming language, an implementation of either a compiler or an interpreter is needed. C and C++ are languages that are compiled from human-readable source into computer-readable object code and then linked into an executable program. Among C++ compilers, the GCC compiler from the GNU Project plays a prominent role because of its distribution policy: it is available free of charge on the Internet. It certainly is the most frequently used compiler in academic and research environments.

Newmat, a matrix class for C++ developed by Robert B. Davies, will also be discussed in this review. Newmat provides various matrix types: regular, symmetric, diagonal, row and column vectors, as well as some more specialized types. Arithmetic operations can be performed using the standard operators `+`, `-` and `*`. Many matrix operations are provided as 'member functions' (a term that will be explained in the section on C++).

This review also presents a small example of an econometric program. We will show a simple OLS program that uses a singular value decomposition.

Finally, it should be noted that this review is not about a commercial package. GCC runs on a far wider variety of platforms than most, if not all, commercial compilers. Newmat can be used with

GCC, and it can also be used with a variety of commercial compilers for PC compatibles and UNIX computers. It will presumably work on Macintosh computers with a C++ compiler too, although confirmation on this is currently not available.

This review is organized as follows. Section 2 introduces some important aspects of C++ programming. The GCC compiler is discussed in section 3. Section 4 presents the newmat matrix class. Section 5 offers an example of an econometric program. Section 6 concludes the review. Finally, details on how to obtain the reviewed software from the Internet are provided in an appendix.

2 C++

The C++ programming language was originally developed at Bell Labs for internal use by AT&T. Stroustrup (1991), who created the language, states in the beginning of the introduction to his book *The C++ Programming Language* that C++ is designed to be a better C, to support data abstraction, and to support object-oriented programming.

Each of these three points will now be considered. Let us first address the C language which was also created at Bell Labs in the early seventies by Dennis Ritchie; see Kernighan and Ritchie (1988) for the language reference. C first gained widespread popularity together with another Bell Labs product, the UNIX operating system, which is written in the C language. Several reasons can be credited for the success of the C language. C allows one to write programs that are highly portable among different operating systems and among different hardware platforms. These programs are also very efficient in terms of performance. Moreover, C is a language that offers low-level interfaces so that it can be used instead of assembly language. But C also provides high level instructions similar to those found in procedural languages like Ada, Fortran or Pascal.

However, the power of C may well be its greatest weakness, because it imposes very few restrictions on what programmers can do. Expressions that are syntactically correct may have semantics that are very different from the programmer's intensions. A classical example is the expression `if (a = b)` which *assigns* the value of b to a and then evaluates the boolean value of this assignment; this always results in a 'true' condition. What is usually meant is `if (a == b)`, a test for equality between a and b. GCC, like most compilers, can be asked to issue a warning in such a situation. Another example of the freedom a programmer enjoys in C is the mild type-checking. C programs can be rather difficult to read, debug and maintain.

C++, on the other hand, overcomes many of the shortcomings of the C language. For example, C++ has more rigid type-checking as a default that can be overridden by explicit casts if needed. Also, function prototypes are required in C++, eliminating the common C mistake of calling a function with an argument the type of which is different from the one declared in the function interface (C compilers can warn about this mistake too). Another improvement of C++ over C is that variables can be declared anywhere in the program. This means that the declaration and the usage of a variable can be brought together, making code more readable. Memory can be allocated via the `new` and `delete` operators as opposed to the `malloc()` and `free()` calls in C. Pointer usage can be replaced by 'references' using the `&` in a similar way as the `var` declaration in Pascal. Input and output can now be handled by streams without the cumbersome formatting strings of the C functions `printf` and `scanf` (they are still available if needed). Thus, one of the biggest assets of C++ is that the language is a superset of the C language: (almost) every C program will compile as a C++ program. It follows that the generality that made C the standard language for development is still available in C++. It also implies that C++ programs can access a large number of routines and libraries available for C.

The second design issue concerns data-abstraction. The 'classical' programming approach is procedural as in Fortran, Algol and Pascal. Procedures are written in order to execute well-defined tasks, and the emphasis lies on the processing. A classical example is a numerical function that calculates and returns the square root of its argument. A corresponding programming paradigm is that of modules. Here, programs are organized in terms of their data rather than in terms of procedures and organized as a set of modules; data can be hidden in these modules. The standard example for a module is a stack where functions for pushing and popping elements are provided. Historically, the next programming paradigm is that of abstract data types. The programmer can extend the set of language-supplied types with her own. Standard examples for user defined types are a point in a Euclidean coordinate system or a two-dimensional array. An abstract data type hides the internal representation of the data in its module. Procedures are provided with the module to operate on the abstract data type. C++ supports this programming style well through its `class` concept.

The third design issue is object-orientation. This is the newest programming paradigm. It addresses shortcomings of abstract data types. Reconsider the example of a stack. An abstract data type for a stack can (when we assume strong type-checking, a clear requirement for safe programming that is met in C++) only deal with one type of data. If a program needs different stacks that operate on different underlying types, very similar code will have to be rewritten for each of these types. An object-oriented language such as C++ offers *templates* to overcome this problem. A stack can then be implemented as a template version and the compiler creates the required incarnations of the stack for the different types from the template version. Barton and Nackman (1994) state three fundamental ideas for object-oriented programming in C++:

1. An *object* is an entity that has both a 'state' because it contains data as well as 'behavior' because it also contains functions that access or modify the data. These functions are called *member functions* to underline the fact that they are part of a given object. An object is created from a `class` by means of a program declaration as any variable is declared with its type.

2. *Class hierarchies* permit reuse of code by a mechanism called inheritance. An example is an abstract base class 'vehicle' that would define certain criteria of a vehicle. Another class 'car' can then inherit all features defined by a 'vehicle' and add those that are specific to a 'car' without having to redefine what is already declared for a vehicle. Next, classes for a 'sedan', 'convertible', 'station wagon', etc., can be derived from that of a car.

3. *Polymorphism* permits one to write routines that can be applied to objects of different types through *virtual functions* in a base class are that redefined in a derived class. The base class merely specifies an interface — the derived class actually fills in the required behavior.

Another important feature of C++ is *operator overloading*. This permits one to assign new behavior to operators that are already present in the C++ language. For example, if `a` and `b` are two objects of a class defining complex numbers that overloads the `+` operator, then the expression `a + b` is valid and creates what we would expect: another complex number that is the result this expression. The newmat matrix class makes use of this feature and provides the operators `+`, `-` and `*` for algebraic operations, the `<<` operator for C++ output via streams, and the operators `|` and `&` for vertical and horizontal concatenation.

The commonly recommended book for C++ is Stroustrup (1991). Some helpful resources are available on the Internet. There is the newsgroup `comp.lang.c++` devoted to the language C++. This newsgroup also has an excellent file that answers frequently asked questions (FAQ) which can be obtained at `ftp://sun.soe.clarkson.edu/pub/C++/FAQ`. This file is itself a thorough introduction to object-oriented programming. Another interesting free document that is available as `ftp://rtfm.mit.edu/pub/usenet/news.answers/C-faq/learn-c-cpp-today` reviews several books and tutorials on C and C++. Some of the material it reviews can be downloaded from the Internet as well. Finally, there is one document that lists free numerical software for C and C++. It can be found at `ftp://ftp.math.psu.edu/pub/FAQ/numcomp-free-c`.

3 GNU GCC

GNU — a recursive acronym for "GNU's not Unix" — is the name for a Unix-compatible operating system being written by the GNU Project of the Free Software Foundation (FSF), a non-profit organization founded by Richard Stallman in 1985. GCC is part of this (not yet complete) operating system. GCC is actually a package of three compilers. First, it is a C compiler for both the often called 'K&R' standard of the first edition of the book by Kernighan and Ritchie as well as the ANSI normed C language of Kernighan and Ritchie (1988). Second, it is a compiler for Objective C, an object-oriented extension to the C language. Third, it is a compiler for C++.

The GCC compiler numbers its versions in the a.b.c form, as do most GNU programs. The first number corresponds to major releases; it was augmented to 2 when the three compilers were merged into one executable. The second number is the minor release, which is incremented whenever new features are introduced. Finally, the third number counts the bug-fix releases. As of June 16, 1995, the current release is 2.7.0. The preceding releases were 2.6.3 and 2.5.8.

GCC is available free of charge via the Internet, as are all the other programs from the GNU Project.[1] GCC runs on almost every version of UNIX, as well as on several other operating systems, such as MVS, VMS and Windows NT. A MS-DOS version exists under the name 'djgpp' and an OS/2 version is available as 'emx'. The appendix contains information about where to find these programs.

We will concentrate on the use of GCC under the Unix operating system. GCC comes with its documentation in the GNU Texinfo format. This Texinfo format can be read on-line through the GNU Info reader or with the GNU Emacs editor. In case more GNU software is installed on a system, it is possibly worth installing one of these two programs because almost all documentation for the GNU Project is distributed in this format. Texinfo files can be converted into TEX files and then compiled and printed or viewed by suitable TEX software. Texinfo can also be converted into the Hypertext Markup Langauage (html) and viewed with, for example, the 'lynx' browser. The installation of GCC is not trivial, but binaries are available for several operating systems.

Once the installation is completed, GCC can be accessed from the command line only. But the error messages and warnings from GCC are much more detailed and helpful than those of several

[1] The complete copyright notice can be found on any of the GNU ftp servers as the file `COPYING`. The GNU copyright is often referred to as the GNU Public License (GPL).

commercial compilers. This makes GCC a fairly productive development tool, especially when integrated with other GNU software. GCC offers numerous command-line switches in order to fine tune its behavior, but only very few are required. As an example, let us consider the following C++ program

```
#include <iostream.h>

int main()
{
  cout << "Hello, World!\n";
}
```

from Stroustrup (1991, p. 23) which can be turned into an executable program via `g++ -o hello hello.cc`.[2] This creates a program `hello` from the source file `hello.cc`. Necessary command-line options will usually be issued from a `Makefile`. These options fall into several categories. 'Overall options' control the gcc output. For example, the `-o` is used for naming the executable. 'C++ language options' and 'C language options' control different language dialect aspects. 'Warning Options' control the amount of messages GCC emits for what could possibly be wrong, misleading or redundant. 'Debugging Options' permit one to determine the amount of debugging information generated. 'Optimization Options' control how much and which kind of code optimization GCC shall attempt. 'Preprocessor Options', 'Linker Options', and 'Assembler Options' control the behavior of the different programs GCC calls in order to create its results. 'Machine Dependent Options' allow one to optimize for different hardware aspects on different computer types. Finally, 'Code Generation Options' provide control for some aspects of how code is created by GCC.

GCC integrates well with other programs from the GNU Project. Running GCC from within the GNU Emacs editor allows for efficient interaction with the GNU debugger GDB. In order for GDB to analyze the program, the debugging switch `-g` must be activated. It is worth noting that GCC is one of very few compilers that allows for simultaneous debugging and optimization options. The debugger can be interfaced from Emacs or through different interface programs that run under the X11 Windows system. Another useful tool is the GNU profiler Gprof. Provided a program is compiled with the `-pg` profiling option, a file `gmon.out` will be created during execution of the program. The profiler analyzes the profiling information from this file and displays usage statistics about the program. Bottlenecks, such as inefficient functions, can be detected to improve the overall performance of the program. GCC can also be used as a cross-compiler so that programs can be created for execution under a different operating system.

In summary, GCC is a very complete compiler for C, Objective C and C++. The interface lacks finesse, but on the other hand the program can be acquired at no cost. Installation can be difficult for the inexperienced user, although this problem is much less severe when a precompiled binary version is available, as is generally the case for the most popular operating systems.

4 NEWMAT

Newmat is a system of matrix classes written by Robert B. Davies; the earliest version dates back to October 1990. It is now in its eighth revision. Davies describes the package as follows in the documentation file `newmata.txt`:

[2] The compiler can also be invoked as `gcc`, but calling it as `g++` is recommended in the documentation. A benefit from calling `g++` is that it links automatically against the C++ library.

> The package is intended for scientists and engineers who need to manipulate a variety of types of matrices using standard matrix operations. Emphasis is on the kind of operations needed in statistical calculations such as least squares, linear equation solve and eigenvalues.

Newmat provides the following C++ matrix classes: `Matrix` for normal rectangular matrices, `UpperTriangularMatrix`, `LowerTriangularMatrix`, `SymmetricMatrix` and `DiagonalMatrix` for memory-efficient storage of these 'special' matrices, as well as `RowVector` and `ColumnVector`. Band matrices are also supported. A variant of the rectangular Matrix, `nricMatrix`, can be used with programs from the book on *Numerical Recipes in C* (Press, Teukolsky, Vetterling, and Flannery, 1992).

Matrix operations are possible using the overloaded algebraic operators `+`, `-` and `*`. Matrices can be concatenated, inverted, transposed, converted between the different types, sorted and printed. Sub-matrices can be extracted. Cholesky decomposition, QR triangularization, singular value decomposition, eigenvalue computation (for symmetric matrices), matrix sorting and fast Fourier transformations are supported. Functions for computing sums and absolute sums of the elements of a matrix, for finding the maximum element, for calculating two different norms, for calculating the trace and for calculating the log of the determinant are provided with newmat. Submatrices can be created and extracted. Matrix dimensions and matrix types can be changed. Using GCC, matrix size is limited to 2 147 483 647 elements as this corresponds to the maximal value of a 32 bit `int`.

Newmat can also be used with the C++ compilers of Borland, Microsoft and Watcom on a PC. On Unix computers, the C++ compilers from AT&T, Sun, and HP can be used instead of GCC. When newmat is used with GCC, all program files have to be renamed because GCC prefers a file suffix of one of the forms `.C`, `.cc` or `.cxx` over the extension `.cpp` that has been chosen by Davies. The makefile performs the required renaming automatically.

Newmat comes with an example file `example.cpp`. It computes a simple linear regression using four different algorithms. The file is extensively commented and serves well as a starting point. The first method is a 'traditional' sum of squares and products method with subtraction of means. A second 'traditional' sum of squares and products method employs Cholesky decomposition. This is followed by the third algorithm which uses QR triangularization; the fourth method uses singular value decomposition. Two more examples are provided: `nl_ex.cpp` is a non-linear least squares example and `sl_ex.cpp` illustrates the use of the equation solver.

Newmat also comes with a comprehensive test program, `tmt`, organized as a series of functions that perform various matrix operations. In total, 365 matrices result from these tests and all of them, with the exception of the very first one, should evaluate to zero. The output of the test program can easily be verified for the desired answer of exactly one non-zero matrix. Memory leakage is also examined. This test program provides a good means of detecting possible problems due to different memory and storage conventions on different computer types.

Documentation for newmat is provided by two formatted ASCII files. The first file, `newmata.txt`, contains seven sections. An introduction to newmat is provided in the first section. It is followed by an explanation of how the user can get started with newmat using one of the different compilers provided. The third section is a reference manual for newmat explaining all of the operators and functions provided by the matrix classes. This reference section is the main part of the documentation. This is followed by a very short section on bugs containing two points related to the exceptions as implemented by Davies. A list of all the files in the package and a form for a problem report constitute the last two sections. The second file, `newmatb.txt`, is more advanced in nature and discusses implementation issues of newmat. It is not necessary to read this file in order to *use* newmat. Given that several files can be opened and read by the user at the same time with today's editors and operating systems, having the documentation as a text file is not inconvenient. Of course, both files can be printed out on about 40 pages.

In a C++ program, a matrix is created by `Matrix A(n,k)` where n and and k are integers specifying the dimensions of the matrix. For any of the symmetric matrices, only one dimension is needed: `DiagonalMatrix D(n)`. Matrices can also be constructed dimensionless provided they are assigned a dimension later.

A nice feature of newmat is the interface to the *Numerical Recipes* by Press, Teukolsky, Vetterling, and Flannery (1992). Using the `nricMatrix` class of newmat, routines from the book can be called as they are printed without having to convert data from one matrix representation to another.

A drawback of working with newmat is the size of the resulting program, because the code from newmat will usually be linked statically into the executable. This can be avoided on an operating system that provides dynamic link libraries — the actual newmat code will then be called on demand from the dynamic link library so that the executable contains only its genuine code. Another weak point is that the user is required to decide once and for all of which type matrices should be composed: `float` or `double`. Given that memory requirements are not as stringent under Unix as they were under other older operating systems, most users of newmat will opt for `double`. A more flexible approach allowing less precise but more memory-efficient representation when needed would be welcomed. As well, a simple `int` array class for reordering indices could be helpful.

Newmat does not offer any routines for reading or writing matrices to or from disks; the user is forced to use her own. Newmat could also be completed by some other functions such as a function to extract the diagonal elements of a matrix into a vector. Some more operators that C++ offers could be overloaded for matrix operations so that, for example, inversion and transposition of a matrix `A` could be commanded by saying `!A` or `~A`, respectively, rather than `A.i()` and `A.t()`.

A mere inconvenience is that Davies prefers the file suffix `.cpp`, a form that GCC does not accept, over, say, `.cc`. The `Makefile` for GNU, `gnu.mak` accounts for that naming difference. Unfortunately, `Makefile` does not come with a target for making a newmat library, but this can easily be added provided the user knows how to do this on her particular system.

5 Example: Ordinary Least Squares

This section presents two files that form a small but complete example for a self-contained OLS program. It is assumed that a file `ols.dat` contains data in ASCII format. The first two entries are interpreted as integers that specify the dimensions n and m of the data matrix. The data matrix itself is supposed to contain n rows where the dependent variable is read from the first column and all explanatory variables from the columns 2 to m. No restrictions are imposed on the size of n or m.

Provided that the include files and the newmat library `libnewmat.a` are installed on the machine such that the compiler and linker find them, the command `g++ -s -O2 -o ols ols.main.cc` ols.example.cc -lnewmat -lm creates an executable named ols from the two source files. The options -s -O2 instruct the compiler to 'strip' the executable program and to optimize the code for size and speed. The options -lnewmat -lm instruct the linker to include code from the newmat library and from the system's math library.

```
1  // ols.main.cc, from Davies' example for newmat
2
3  #define WANT_STREAM             // include.h will get stream fns
4  #define WANT_MATH               // include.h will get math fns
5                                  // newmatap.h will get include.h
6  #include "newmatap.h"           // need matrix applications
7  #include "newmatio.h"           // need matrix output routines
8
9  void ReadMatrix(ColumnVector& Y, Matrix& X, const char* file);
10 void OLS(const ColumnVector& Y, const Matrix& X,
11          ColumnVector& Beta, ColumnVector& StdErr,
12          ColumnVector& Fitted, ColumnVector& Residual, DiagonalMatrix& Hat);
13
14 void main()
15 {
16   ColumnVector Y, Beta, StErr, Fitted, Residual;
17   Matrix X;
18   DiagonalMatrix Hat;
19
20   ReadMatrix(Y, X, "ols.dat");
21   OLS(Y, X, Beta, StErr, Fitted, Residual, Hat);
22
23   cout << "\nEstimates and their standard errors\n\n";
24   cout << setw(11) << setprecision(5) << (Beta | StErr) << endl;
25   cout << "\nObservations, fitted value, residual value, hat value\n";
26   cout << setw(9) << setprecision(3) <<
27       (Y | X | Fitted | Residual | Hat.AsColumn());
28   cout << "\n\n";
29   return 0;
30 }
```

The effect of the two #define statements in lines 3 and 4 is that newmat will load the 'include files' iostream.h and math.h that are required for stream output and mathematical functions, respectively. Lines 6 and 7 load the 'include files' that declare newmat applications and input/output operations. Lines 9 to 12 declare the prototypes of the two functions ReadMatrix() and OLS(). The sole function in this file is main(), the function that constitutes a C or C++ program. In this simple program, we first declare all the required variables. The function ReadMatrix() is called in order to read the contents of the two variables Y and X from the file that we indicated. Note that we do not have to specify the dimensions of these any of these variables, see below for how ReadMatrix() implements this. Next we call the OLS function and then print the results onto the standard output, usually the screen provided the program's output has not been redirected to a file. Note how in these output lines we use C++ streams formatting and matrix concatenation using the operators << and | that are overloaded by newmat.

The following file completes the example. It contains the actual OLS function, derived from the fourth example provided by Davies with newmat, and a simple routine to read a matrix from a file.

```
#include <fstream.h>           // include declarations for file streams
```

```
 2
 3 #define WANT_STREAM                  // include.h will get stream fns
 4 #define WANT_MATH                    // include.h will get math fns
 5                                      // newmatap.h will get include.h
 6 #include "newmatap.h"                // need matrix applications
 7 #include "newmatio.h"                // need matrix output routines
 8
 9 void OLS(const ColumnVector& Y, const Matrix& X,
10          ColumnVector& Beta, ColumnVector& StdErr,
11          ColumnVector& Fitted, ColumnVector& Residual, DiagonalMatrix& Hat)
12 {
13   int nobs = Y.Nrows();
14   int nvar = X.Ncols();
15   Matrix U, V;
16   DiagonalMatrix D;
17   SVD(X,D,U,V);                      // X = U * D * V.t()
18   Fitted = U.t() * Y;
19   Beta = V * ( D.i() * Fitted );
20   Fitted = U * Fitted;
21   Residual = Y - Fitted;
22   double resvar = Residual.SumSquare() / (nobs-nvar);
23   // get variances of estimates
24   D << (V * (D * D).i() * V.t())*resvar;
25   StdErr.ReDimension(nvar);
26   for (int i=1; i<=nvar; i++)
27     StdErr(i) = sqrt(D(i));
28   Hat << U * U.t();
29 }
30
31 void ReadMatrix(ColumnVector& Y, Matrix& X, const char* file)
32 {
33   ifstream data(file);          // open input file stream 'data' from 'file'
34   int n, k;                     // rows and columns of data
35   data >> n >> k;               // read matrix dimensions from data
36   Matrix D(n,k);                // allocate data matrix
37   for (int i=1; i<=n; i++)      // read n*k values
38     for (int j=1; j<=k; j++)
39       data >> D(i,j);
40   Y = D.Column(1);              // load first column into Y
41   X = D.Columns(2,k);           // load remaining columns into X
42 }
```

Lines 9 to 11 define the interface of the OLS function. A column vector Y and a matrix X are passed as constant references. This means that they are efficiently passed to the function by a reference using `&` but that the underlying data may not be changed because of the `const` attribute. Thus the data in Y and X can only be returned unaltered. The OLS function also returns five result vectors that contain the estimates, their standard errors, the fitted values, the residuals and the diagonal elements of the hat matrix. Next, two local variables are set to the number of observations and the number of variables. Note that the objects themselves provide this information through member functions. After the declaration of the required matrices U, V and D, the design matrix X is decomposed by a call to the `SVD()` provided by newmat. Lines 18 to 21 then determine the parameter coefficients, the fitted values and the residuals. In line 22, an estimate of the variance of the error terms is obtained through a call to the member function that returns the sum of squared elements of the object it is applied to; this result is divided by the degrees of freedom. In line 24, the diagonal matrix D is assigned the diagonal elements of the covariance matrix. This assignment is done via the overloaded `<<` operator because it is an information-losing operation as we do not want to keep the off-diagonal elements of the covariance matrix. The standard errors are then computed in a loop over all the elements of D since there is no operator or function in newmat to do this in one single operation.

In the function `ReadMatrix()`, an input file stream is opened from the given filename in line 33. Two integer values n and m are read first; these define the size of the data matrix. We then simply loop over $n \cdot m$ elements that are read individually. Finally, the first column of the data matrix is assigned to the column vector Y and the remaining $m-1$ column of the data matrix D are assigned to the matrix X. This routine is neither efficient, as it does not buffer the input stream, nor safe, as no error checking is done. However, these features can easily be added; see Stroustrup (1991, chapter 10) for details.

6 Conclusion

This review has discussed some aspects of C++ programming in an econometric context. The possible rewards can only be seen with medium to large scale projects — for running only a few standard regressions, the user is of course advised to use standard econometric packages. However, given that econometric research frequently involves new methodologies that require programming, a careful approach based on existing modules, as for example the newmat matrix class, can save a great deal of programming time and effort.

References

Barton, John J. and Lee R. Nackman (1994). *Scientific and Engineering C++: An Introduction with Advanced Techniques and Examples*. Reading, Mass.: Addison-Wesley. The source code for this book is available at `ftp://software.watson.ibm.com/pub/SciEng`.

Kernighan, Brian W. and Dennis Ritchie (1988). *The C Programming Language* (Second ed.). Englewood Cliffs, N.J.: Prentice Hall.

Press, William H., Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery (1992). *Numerical Recipes in C: The Art of Scientific Computing* (Second ed.). Cambridge: Cambridge University Press.

Stroustrup, Bjarne (1991). *The C++ Programming Language* (Second ed.). Reading, Mass.: Addison-Wesley.

Appendix: Ftp sites for the software reviewed

The official Internet host for the GNU Project is `ftp://prep.ai.mit.edu/pub/gnu`. It contains GCC as well as all GNU programs and documentation. The following list indicates ftp sites that mirror this host.

Asia:
ftp.cs.titech.ac.jp
utsun.s.u-tokyo.ac.jp/ftpsync/prep
cair-archive.kaist.ac.kr/pub/gnu
ftp.nectec.or.th/pub/mirrors/gnu

Australia:
archie.au/gnu
(archie.oz or archie.oz.au for ACSnet)

Africa:
ftp.sun.ac.za/pub/gnu

Middle-East:
ftp.technion.ac.il/pub/unsupported/gnu

Europe:
irisa.irisa.fr/pub/gnu
ftp.univ-lyon1.frpub/gnu
ftp.mcc.ac.uk
unix.hensa.ac.uk/mirrors/uunet/systems/gnu
src.doc.ic.ac.uk/gnu
ftp.ieunet.iepub/gnu
ftp.eunet.ch
nic.switch.ch/mirror/gnu
ftp.informatik.rwth-aachen.de/pub/gnu
ftp.informatik.tu-muenchen.de
ftp.win.tue.nl/pub/gnu
ftp.funet.fi/pub/gnu
ftp.denet.dk
ftp.stacken.kth.se
isy.liu.se
ftp.luth.se/pub/unix/gnu
ftp.sunet.se/pub/gnu
archive.eu.net

South America:
ftp.inf.utfsm.cl/pub/gnu
ftp.unicamp.br/pub/gnu

Western Canada:
ftp.cs.ubc.ca
/mirror2/gnu

USA:
wuarchive.wustl.edu/systems/gnu
labrea.stanford.edu
ftp.digex.net/pub/gnu
ftp.kpc.com/pub/mirror/gnu
f.ms.uky.edu/pub3/gnu
jaguar.utah.edu/gnustuff
ftp.hawaii.edu/mirrors/gnu
vixen.cso.uiuc.edu/gnu
mrcnext.cso.uiuc.edu/pub/gnu
ftp.cs.columbia.edu/archives/gnu/prep
col.hp.com/mirrors/gnu
gatekeeper.dec.com/pub/GNU
ftp.uu.net/systems/gnu

The 'djgpp' port of GCC 2.6.0 for MS-DOS by D.J. Delorie can be obtained on the Internet at ftp://oak.oakland.edu/SimTel/vendors/djgpp. It comprises a 32-bit i386 DOS extender with symbolic debugger, development libraries, ports of GNU Bison, Flex, GAS and Binutils and comes with full sourcecode. It supports the memory models XMS, VDISK, VCPI and DPMI.

The 'emx' port of GCC 2.6.3 for OS/2 2.x and OS/2 Warp by Eberhard Mattes can be obtained from ftp://ftp-os2.cdrom.com/pub/os2/unix/emx09a. It comprises the GNU assembler, debugger, linker and some other utilities, documentation, OS/2-specific BSD C libraries, and many Unix-related library functions like 'fork'. Programs compiled by 'emx' can also run on a 80386 under DOS.

Newmat can be obtained from ftp://oak.oakland.edu/SimTel/msdos/cpluspls/newmat08.zip and ftp://wuarchive.wustl.edu/usenet/comp.sources.misc. It is also available via ftp from the CodEc archives at netec.mcc.ac.uk and netec.wustl.edu in /pub/NetEc/CodEc/C++.