Econometrics with Octave

Dirk Eddelbüttel*
Bank of Montreal, Toronto, Canada. Dirk.Eddelbuettel@bmo.com

November 1999

Summary

GNU Octave is an open-source implementation of a (mostly Matlab compatible) high-level language for numerical computations. This review briefly introduces Octave, discusses applications of Octave in an econometric context, and illustrates how to extend Octave with user-supplied C++ code. Several examples are provided.

1 Introduction

Econometricians sweat linear algebra. Be it for linear or non-linear problems of estimation or inference, matrix algebra is a natural way of expressing these problems on paper. However, when it comes to writing computer programs to either implement tried and tested econometric procedures, or to research and prototype new routines, programming languages such as C or Fortran are more of a burden than an aid. Having to enhance the language by supplying functions for even the most primitive operations such as transposing a matrix adds extra programming effort, introduces new points of failure, and moves the level of abstraction further away from the elegant mathematical expressions. As Eddelbüttel (1996) argues, object-oriented programming provides 'a step up' from Fortran or C by enabling the programmer to seamlessly add new data types such as matrices, along with operations on these new data types, to the language. But with Moore's Law still being validated by ever and ever faster processors, and, hence, ever increasing computational power, the prime reason for using compiled code, *i.e.* speed, becomes less relevant. Hence the growing popularity of interpreted programming languages, both, in general, as witnessed by the surge in popularity of the general-purpose programming languages Perl and Python and, in particular, for numerical applications with strong emphasis on matrix calculus where languages such as Gauss, Matlab, Ox, R, Splus, which were reviewed by Cribari-Neto and Jensen (1997), Cribari-Neto (1997) and Cribari-Neto and Zarkos (1999), have become popular. This article introduces another interpreted language focussed on numerical applications: Octave.

Octave is a program, as well as a programming language, for doing numerical work in a convenient yet expressive and powerful fashion. Octave was written primarily by John W. Eaton of the University of Wisconsin-Madison with various smaller contributions from other programmers across the Internet. The history of Octave can be traced back (as far as 1988) to the idea of providing companion software

*Comments and suggestions by Francisco Cribari-Nero, John Eaton, Kurt Hornik, James MacKinnon, Lisa Powell and Colin Telmer are gratefully acknowledged.

for a textbook in chemical engineering, written by two researchers from the University of Wisconsin-Madison and the University of Texas. Full-time development for Octave began in the spring of 1992. Version 1.0 was released in February of 1994, and version 2.0 followed in January of 1997. Octave is part of the GNU project (which is on the Web at `http://www.gnu.org`) and released under the GNU General Public License (GPL). It can therefore be used, studied, extended or simply copied just like any other Free Software program.[1]

Econometricians might want to employ Octave for a variety of applications as it provides a convenient tool for interactive work which can range from simple applications to ad hoc simulations. The user interface, *i.e.* the Octave shell, is flexible and powerful. The rich set of mathematical operators and functions make Octave a very good prototyping and programming platform, or data pre- and post-processing application. As Octave can be extended with new C, C++ or Fortran code that runs at the speed of compiled code, it is a good building block for Monte Carlo simulations. Octave is very similar to Matlab, and to a lesser extent to Gauss. However, both of these are commercial 'closed source' programs whereas Octave, as noted above, is distributed under the GNU GPL.

Octave is currently at version 2.0.14. Akin to Linux kernel development, the 'stable' 2.0.* branch corresponds to the released version. Meanwhile, development progresses with a 2.1.* branch (currently at 2.1.14) which will eventually replace the 2.0.* series. While this review focuses on Octave 2.0.14, it should also be relevant to the 2.1.* series. The Octave web site at `http://www.che.wisc.edu/octave` is the best starting point for Octave. It contains the source code as well as binary versions for different operating systems. Moreover, documentation is available in several forms of the manual as well as via the mailing list archives which contain a large number of helpful posts. The `help-octave` list is for general discussions related to installation and use of Octave and the `bug-octave` list is for reporting bugs (preferably using the `bug_report` command). It should be pointed out that John Eaton, on several occasions, has posted patches within hours of a bug report — an exceptional service. Also of note are the `octave-sources` list which distributes contributed functions and scripts, and the `info-octave` list for announcements. Octave is also included in several Linux distributions.

The rest of this review is organized as follows. The next section provides an overview of Octave, followed by a critical examination of its strengths and weaknesses. Sections 4 and 5 provide a simple Monte Carlo experiment and an example of extending Octave with C++ code before section 6 concludes.

2 Overview

This section introduces core aspects of the Octave language. Certain features will be familiar to readers who have worked with other matrix-type language such as Matlab, Gauss or Ox. Users of Matlab in particular should have no problems using Octave.

Octave has a few standard data types. Without doubt, the most important one is the **matrix**, which also covers the scalar and vector cases. Matrices support real and complex arithmetic. For example, `A = [1, 2, 3; 4, 5, 6]` assigns a 2 by 3 matrix to the variable A. Note that the semicolon is used to separate rows, whereas the comma separates elements within a row. The result of this operation, *i.e.* the matrix A, will be displayed in an interactive session as the assignment is not terminated by a

[1] Several terms such as 'copylefted', 'Free Software' and 'Open Source' are used to describe software which is available in source code format, and without restrictions on its use. Prominent examples include software from the GNU project, the X11 window system and the Linux operating system. See Eddelbüttel (1996) for a review of the GNU GCC compiler in conjunction with a C++ matrix class, and MacKinnon (1999) for a review of the Debian GNU/Linux operating system.

semicolon. Conversely, in `d = det(A*A');`, the scalar d that results from computing the determinant of AA', will not be displayed because this assignment is terminated by a semicolon. Note that we did not have to declare either of these variables; their type is determined at run-time. Octave allocates memory as needed, which also allows for 're-allocating' a variable to a different type, and for 'growing' variables by extending the array dimension. In passing, we also introduced the operator `'` for transposing a matrix.

Another important data type is the **range** which is defined by using a colon between the lower and upper bound. For example, `1:10` defines the range of integers from 1 to 10, and returns it as a row vector. An increment can be added as an optional second argument: `(10:5:30)'` generates a column vector with the six elements $5, 10, 15, 20, 25, 30$. Related to ranges are **indices**. Suppose we write `T=1:100;`. Then `T(5)` selects the fifth element, and `T([10, 20, 40])` returns a vector which contains the elements of T with indices 10, 20 and 40. Indexing expressions such as the last one can appear on the left-hand side as well as on the right-hand side of an assignment. An important operator for indices is the single colon which could loosely be called an 'all' operator. For example, `A(:,2)` refers to the second column as all rows are selected via the colon.

Character **strings** are also available. A string is a row vector of characters: `a = "hello"` assigns a string to a variable a. We can access elements of a just like we can access numerical matrices: `a(3)` selects the first 'l'. Further, matrix operations can be used for concatenation: `b = [a, " world"]` creates one longer string (row vector). A semicolon can be used to separate rows just as with numerical matrices: `c=["one"; "two";"three"];` creates a matrix of three rows, and five columns as Octave maintains these 'string vectors', *i.e.* 'character matrices', in rectangular form; the first two rows are automatically padded with blanks.

Lastly, Octave supports **structures**. For example, `a.b = c;` assigns the content of the variable c to the element b of a. Therefore, a is now of type structure. Structures can be recursive: the variable c could itself have been a structure. This provides a simple way to create custom types to regroup information in a simple entity in order to maintain compact interfaces between functions. For example, a function for linear regression can return a structure with the usual results: a vector with parameter estimates, another vector with the residuals, another vector with the fitted values, plus scalars for various diagnostics statistics. All of these could be returned individually, but it is clearly more appropriate to regroup them in one structure. Next, one could write a function (with the regression result structure as the sole argument) to summarize the regression result.

Operators work as expected in Octave. The language has the usual algebraic, boolean and comparison operators. For several algebraic operations, Octave also has 'dot' operators that apply to individual elements. To use these, the dot is prepended to the operator. For example, if A and B are two matrices, then `A*B` multiplies them in the habitual sense of matrix multiplication whereas `A.*B` computes the Hadamard product obtained by elementwise multiplication. This also extends, where appropriate, to other operations. For example, in `1.05.^(1:30)`, the scalar value 1.05 is raised to the power of each element of the range 1:30 thereby creating a vector of the first 30 powers of 1.05. Another example is the combined use of boolean operators and indexing. Suppose that the matrix D contains time-series data, and the first column provides the date for each observation in a format like 19990811. One can then simply extract all observations matching a date range. For example, `DD=D(D(:,1)>=19800101 & D(:,1)<=19891231, :)` extracts all columns of the rows corresponding to the 1980s and assigns them to DD.

Expressions are central to Octave statements. Each expression evaluates to a value, which is printed if no semicolon follows the expression. The result of an expression can be tested, stored, passed as

an argument to a function or assigned to a variable. Statements can contain several expressions. For example, an indexing expression might determine which elements of the left-hand side are assigned the values of the right-hand side expression as `A([1, 3]) = [2, 6];`. Here, the matrix expression `[1, 3]` determines that the first and third elements of A are assigned the values on the right. The right-hand side contains a matrix of matching dimension; each of the two (here constant) elements could itself be a more complicated expression.

Octave **keywords** are similar to those in other procedural languages. Conditional execution can be coded using `if` *(condition)* `then` *code* `endif` with an optional `else` branch. There is also a `switch` statement with mutually exclusive `case` branches. Two types of loops are available: `for` i = *index expression code* `endfor`, as well as `while` *(condition) code* `endwhile`. In all these cases, the shorter keyword `end` may be substituted for the longer and more explicit versions shown here. Control flow can be altered with the `break` and `continue` keywords. Octave also has limited support for exceptions using `unwind_protect` and `unwind_protect_cleanup` as well as `try` and `catch`. Lastly, file input and output operations are available for ASCII and binary format via the `load` and `save` keywords.

Functions are probably the most central part in using Octave for anything more than simple ad-hoc operations. Octave provides many numerical commands as builtin commands. On a system built with the 'enable-shared' configuration option, these correspond to dynamically-linked code which is loaded as needed. Otherwise, all these commands are built into the main Octave binary.[2] Most of the builtin numerical Octave commands come from well-known **libraries** from the Netlib repository such as Linpack, Minpack, Lapack, Randlib, Balgen, Dassl or Odepack. Not only is this a good idea in terms of code re-usage, but it is also reassuring in light of the findings by McCullough (1999) about numerical reliability, or the lack thereof. By tying Octave to these tried-and-tested numerical 'engines', the benefit of man years of code testing, debugging and improvements are reaped, and the end user is much less likely to be bitten by deeply hidden numerical bugs.

Beyond these core commands, Octave 2.0.14 comes with over 400 function text files. These functions cover various topic areas: from linear algebra functions such as `kron` for a Kronecker product, trigonometric functions such as `asech` for the inverse hyperbolic secant, general functions such as `reshape` or `shift` to alter matrices, miscellaneous functions such as `bincoeff` to functions for polynomials, special matrices such as `toeplitz` or `hilb` or string functions such as `substr` or `index`. A large number of statistical functions, often contributed by Kurt Hornik and his colleagues from the Technische Universität Wien are included. These cover functions for the cdf, pdf, inverse or random numbers for 22 distributions, as well as functions to compute descriptive statistics, moments, ranges, qqplots or classic statistical tests. A number of these statistical routines were previously distributed separately by Hornik et al. under the name `octave-ci`, but with Octave release 2.0.14, these were integrated into the main Octave release. Some useful files remain in the `octave-ci` package, notably `aload` and `asave` for a very general interface to reading and writing data from ASCII files. The package is available from `ftp://ftp.ci.tuwien.ac.at/pub/octave/octave-ci.tar.gz`, and is also included in the Debian GNU/Linux distribution which was reviewed by MacKinnon (1999).

Functions are the standard way of adding another command: one simply creates a file (in a directory that is known to Octave, which could be the current directory, or any directory pointed to from the `LOADPATH` variable) with the same name as the command itself, followed by the suffix `.m`. The

[2]The availability of this 'enable-shared' option depends on the operating system. Essentially all Unix and Linux versions provide it; but it is still unavailable on Windows 95/98/NT. Systems without this option cannot be extended incrementally as shown in the fifth section. Rather, the entire Octave interpreter has to be rebuilt to incorporate an addition.

following example provides a trivial function returning the current date as a numeric variable in the YYYYMMDD format.

```
1  ## Usage:  date = today ()
2  ## Returns today's date in YYYYMMDD format.
3  function date = today ()
4    date = str2num(strftime("%Y%m%d", localtime(time())));
5  endfunction
```

As can be seen from the example, a function is defined by the keywords `function` and `endfunction`. This particular function `today` returns a single variable, and has no arguments. The example also shows that Octave can access system library functions such as `time`, `localtime` and `strftime`. The comments in the first two lines supply the help text which is displayed if the user requests `help today`. Functions can be created 'on the fly' in an interactive Octave session, but extra steps must be taken to save them for another session (beyond the save via command history).

Octave also contains **plotting** commands which are implemented as wrappers around the external Gnuplot program. These commands include a generic `plot` command for plotting one or several series, as well as commands for plotting histograms and bar charts. All of these can be annotated, and it is possible to have several plot windows open at the same time (which requires a Gnuplot binary more recent than the 3.5 releases). The Octave plot commands are very convenient, but clearly not as complete as the Matlab plotting functions and hence might not satisfy all needs for publication-quality graphs. However, add-on packages have been provided by Octave users. Debian GNU/Linux provides additional graphics packages such as `octave-plplot`, an Octave interface to the PLPlot program, and `octave-epstk`, a package for the direct creation of encapsulated postscript graphs.

3 Strengths and Weaknesses

Octave is a very convenient tool for numerical and data-centered work. Most importantly, the Octave language, and its commands, work in a very intuitive way. In the following example, a data set of IBM stock prices (with the dates in the first column) is loaded from an ASCII file using the `load` command. Log-returns are calculated by computing the differences between the logarithms of the stock prices (given in the second column); this takes just one line invoking the `diff` and `log` commands. Next, descriptive statistics are computed using the `statistics` function (which returns the minimum, first, second and third quartiles, maximum, and the first four moments), and displayed as there is no semicolon. We then compute a one-sided t-test for the null of positive returns. Last, the returns are plotted as a time series and, in a second window, the histogram is displayed as well. As we are using explicitly numbered plot windows, both graphs will be displayed simultaneously.

```
1  IBM = load IBM.dat
2  IBMret = diff(log(IBM(:,2)));
3  statistics(IBMret)
4  t_test(IBMret, 0, ">")
5  figure(1)
6  plot(IBMret)
7  figure(2)
8  hist(IBMret,50)
```

Octave is also very flexible in its communication with the host operating system. The `system` and `popen` commands allow the programmer to execute other programs or scripts, and to recover results from launching these programs. A trivial example would be `loadavg=str2num(system("w | head`

`-1 | cut -c 52-56"));` which, in a rather brutish way, extracts the current system load from the `w` command and filters the current load out of the returned text. A fine example is provided by the `aload` script from the `octave-ci` package. This shows how to wrap Octave around calls to the Unix tools `awk` and `sed` so that reading data from a comma-delimited file 'data.csv' becomes as simple as `X = aload("data.csv", Inf, Inf, ",");`. This specifies that an unlimited number of rows and columns shall be read from the file (provided Octave can allocate enough memory to accommodate the finite amount of data in the file) and selects the comma as the data delimiter. Also, shell-like commands such as `ls`, `dir`, `pwd` or `cd` are available directly in Octave.

Another strength of Octave for interactive work is due to the use of the GNU Readline library. Not only does this add a 'browseable' command-line history which can be accessed with cursor keys, or standard Emacs key bindings as CTRL-n and CTRL-p for the next and previous line, it also saves the command history between sessions, allowing the user to easily continue where she left off. The command-line history can also be searched using CTRL-r. Another gem is the command-completion. As an example, to compute a Kolmogorov-Smirnow test, one simply types `kol` followed by a TAB which calls for an automatic expansion to the matching command, or commands in case of an ambiguity. In this case, the command-line is expanded to `kolmogorov_smirnov_`, the unambiguous expansion. Also shown are the three commands that match this expansion: `kolmogorov_smirnov_cdf`, `kolmogorov_smirnov_test`, and `kolmogorov_smirnov_test_2` for the cdf and test variants against the Normal and a specified alternative dataset. Finally, the interactive mode also provides the help command. It can be used to look up the help provided with a command or function as for example in `help(t_test)`. It can also be used to look up the Octave manual by adding the `-i` option as in `help -i load`.

On the topic of user interfaces, the Emacs support deserves special mention. Three Emacs lisp files, written by Kurt Hornik based on prior work by John Eaton, provide modes to write Octave code, to run Octave in an Emacs buffer (with more scrollable history than in a terminal window) and to access the help system. This Emacs mode supports standard editor features such as automatic colour highlighting and indentation, as well as more specific features. One useful example is the stanza for new Octave files. As noted above, creating a new file is the first step in writing new Octave functions. With the Emacs mode, pressing 'CTRL-c f' in a new and empty file buffer (whose name should end in `.m` to assign Octave mode) invokes an Emacs function to write the basic structure of the function. It prompts for the function name, inferring a default name from the buffer name, and prompts quotes for function arguments and return values. These are then inserted into the hitherto empty file, along with a stanza for the help code. Similarly, the use of the abbrev-mode reduces the amount of typing: for example `'w` triggers insertion of `while ()`. Other notable features are the invocation of context-sensitive help, and the debugging features which allow one to send individual commands, or blocks of commands to the Octave interpreter. All of these make for a rather nice development platform.

A key advantage of Octave is the 'copylefted' availability of the source code. Usage of Octave is therefore not bound to a particular machine or license, and Octave, having been ported to most common hardware platforms, can be installed throughout a department, lab or university. Octave was, and is, developed primarily under Unix, so it is not surprising that it has been compiled on most, if not all, major flavours of Unix. Octave has also been ported to Windows 95/98/NT, using the freely available CygWin tools (see `http://sourceware.cygnus.com/cygwin/`) but remains a little more difficult to install as a binary package than under Linux. Compiling and installing Octave from source is very straightforward and follows the usual cycle for a GNU program: `configure; make; make install` (but one might want to select some configuration options such as `--enable-shared` and `--enable-lite-kernel`). Also provided is a target `make test` which runs a very comprehensive set of regression tests to ensure that Octave has been properly built.

Octave is certainly close to Matlab. John Eaton aims for Matlab compatibility whenever this is sensible; however, in a few cases implementation details differ. These small differences add an extra burden to the creation of code that is intended to run on both platforms. It should be possible to port most code written for Matlab to Octave with some extra work, provided no Matlab-specific tool boxes, or new features from Matlab5, were used.

From an econometrician's viewpoint, the obvious weakness of Octave is a lack of actual econometrics code — but this is something that the econometrics community could address by making routines and procedures available on the Internet. A few estimation routines are available; for example, one for nonlinear regression (at `ftp://fly.cnuce.cnr.it/pub/software/octave/`) which was ported from a publicly available Matlab routine. Generally speaking, not even the (public domain) Matlab archives from the Mathworks (at `http://www.mathworks.com/support/downloads.shtml`) and the Mathtools (at `http://www.mathtools.net`) have much specific econometrics code. One example of an econometric code archive for Matlab is `http://www.econ.utoledo.edu/matlab_gallery`.

One of the key problems in adding more econometrics code, in particular for non-linear estimation, lies in the requirement for an adequate non-linear (constrained) optimization routine. Unfortunately, most of the code available to the research community has been released under somewhat ill-defined licenses (as for example a simple 'free for research but contact author for other uses'). This is fundamentally in conflict with the GNU GPL (as it restricts usage of the code), and therefore precludes the addition of code thus released to GNU projects such as Octave. A notable exception is a package for semidefinite programming, written by Lieven Vandenberghe and Stephen Boyd and ported to Octave by A. Scottedward Hodel (who is also the author of substantial control theory package available at `ftp://ftp.eng.auburn.edu/pub/hodel/`); it is also available as a Debian package.

Octave is a very complete program and environment, despite the noticeable lack of actual econometrics code. Unfortunately, the documentation is less complete. This is, however, understandable. After all, the code has been produced mostly by volunteers who prefer to concentrate on coding, rather than something less thrilling such as writing documentation. In effect, the existing documentation is actually very good. As is the case with other programs from the GNU project, the documentation is provided in a meta-format called Texinfo. From this format, versions for local on-line reading and searching (Info), printing (TEX, PostScript, PDF) and local or remote web-browsing (HTML) can be derived. The (postscript version of the) main Octave manual prints to over 250 pages and its HTML version can be read on-line at `http://www.che.wisc.edu/octave/doc/octave_toc.html`. It provides a good starting point, as well as a reference for Octave. Unfortunately, documentation on how to use, or re-use, Octave internals and libraries is missing. Also missing is information about the C++ classes, which makes writing C++ code to extend Octave more difficult than it should be. In this case, the `help-octave` mailing list is the best route for those seeking help.

4 A simple Monte Carlo example and comparative performance

In this section, an example of a simple Monte Carlo experiment, taken from Cribari-Neto and Zarkos (1999), is shown. Examples such as this are valuable for teaching purposes as they encapsulate the essential features of a Monte Carlo experiment in a short segment of code. Data generation, simulation, analysis and presentation of results can all be expressed in a few lines. As in the cited source, the actual computation of the r Monte Carlo replications is collapsed into the execution of just *one* command, showing how to vectorize an entire simulation. This has obvious advantages for speed. However, more complex problems must typically be dealt with by writing loops:

```
1  function MC = mcsim (r)
2    if (nargin < 1)
3      r = 1000;
4    end
5    if (r <= 0)
6      error("The number of replications must be positive.\n");
7    end
8    Beta = [ 7.3832; 0.2323 ];    # true parameters
9    sigma2 = 46.852;              # true variance
10   x = [ 25.83; 34.31; 42.5; 46.75; 48.29; 48.77; 49.65; 51.94;
11        54.33; 54.87; 56.46; 58.83; 59.13; 60.73; 61.12; 63.1;
12        65.96; 66.4; 70.42; 70.48; 71.98; 72; 72.23; 72.23;
13        73.44; 74.25; 74.77; 76.93; 81.02; 81.85; 82.56; 83.33;
14        83.4; 91.81; 91.81; 92.96; 95.17; 101.4; 114.13; 115.46 ];
15   t = length(x);
16   X = [ ones(t,1), x ];
17   Ysim = (X*Beta)*ones(1,r) + randn(t,r)*sqrt(sigma2);
18   MC = inv(X'*X) * X' * Ysim;   # returns estimates on all r replications
19   title("Histogram of b2");
20   xlabel("B2");
21   ylabel("Frequency");
22   hist(MC(2,:), r^0.4);
23 endfunction
```

Lines 2 to 4 add a test and an assignment to provide a default number of replications. A rather useful feature of the S language allows specifying such a default in the function header. Presumably, this could only be added to Octave by losing even more Matlab compatibility. Lines 5 to 7 test to ensure that the parameter is sensible. Lines 8 to 15 set the true parameters, and assign the invariant explanatory variables. Line 16 generates a t by r matrix — in other words, the t observations on $X\beta$ are repeated r times, and standard-normal error terms are added. The core of the simulation is line 18 where the usual least-squares estimation is applied simultaneously to all r simulation steps. Lines 19 to 22 generate an annotated histogram. Due to the memory requirements from storing r by t matrices, it may be infeasible to use very large r. The alternative is to use a loop.

```
1    MC = zeros(2,r);
2    M = inv(X'*X) * X';
3    for i = 1:r
4      Ysim = X*Beta + randn(t,1)*sqrt(sigma2);
5      MC(:,i) = M * Ysim;
6    end
```

Here, we first allocate $2r$ elements for the result matrix which, while not required, enhances performance as Octave will not have to resize the matrix during the course of the simulation. We then calculate the matrix M before the actual loop calculates the r Monte Carlo replications each of which consists of the data-generation, followed by the linear estimation.

This example also provides a good test-bench for performance comparisons. We will time and compare Octave to two other programs / languages. The first is another product of the GNU project: R, the open-source implementation of the statistical language S. The second is the commercial closed-source Matlab package, a very powerful (and expensive) language cum environment very similar to Octave, but equipped with built-in editors, comprehensive graphics and optional tool boxes. This ties the comparison to the one by Cribari-Neto and Zarkos (1999) who compared R to the commercial package Splus. We ran the vectorized version as well as the explicit loops version at four different experiment sizes. The results are summarized in Table 1.

Table 1: Execution times for Monte Carlo example on Linux

	Vectorized				Loop			
r	Octave	Oct/MT	R	Matlab	Octave	Oct/MT	R	Matlab
1000	0.18	0.10	0.09	0.08	0.52	0.35	0.53	0.18
5000	0.83	0.40	0.43	0.24	2.43	1.66	2.58	0.74
10000	1.63	0.79	0.84	0.44	4.85	3.30	5.12	1.44
50000	8.10	3.88	4.94	2.12	23.35	16.70	25.75	7.14

Note: Execution time is in seconds; each experiment was run five times. The average of the last three runs, as reported by `cputime()` and `system.time()`, respectively, is shown. All simulations were run on a 333 MHz laptop with 96 MB of memory running Linux kernel 2.2.10. The Octave version was 2.0.14, the R version was 0.64.1 and the Matlab version was 5.3.0.108. Oct/MT refers to Octave with the MT random number generator (see next section). R required an explicit allocation of half the computer's memory to run the largest simulation.

This simple comparison allows for several interesting observations. Clearly, R is faster than Octave on the vectorized example where it takes about half as much time as Octave. On the other hand, Octave is faster than R on the loop example which is more reminiscent of a real-life Monte Carlo study. These comparisons have to be qualified on two counts. First, R contains an integrated graphics engine whereas Octave has to communicate with Gnuplot (through Unix pipes, not files). This communication with an external program must slow Octave down. Second, Octave uses the Randlib random number generator by Brown, Lovato, Russell, and Venier (1997) which is statistically sound, yet 'expensive' due to an underlying system of 32 virtual random number generators each of which provides 1,048,576 blocks of numbers where each block is of length 1,073,741,824. This is packaged such that a single generator with period 2.3×10^{18} is seen if the distinct blocks are not required. We have replaced the call to `randn` in the example above with a call to the much faster Mersenne Twister random number generator by Matsumoto and Nishimura (1998), coupled with a standard routine for converting uniform random deviates into standard-normal ones (see the next section). The results are shown in Table 1 under the 'Oct/MT' heading and are quite striking: on the vectorized example, Octave now requires less than half the time and clearly outperforms R. Similarly, Octave dominates R further on the loop example. This indicates that the underlying numerical operations in Octave are indeed fast.

For completeness, we also include the performance of Matlab, a commercial closed-source package very similar to Octave. The performance seems to warrant the significant price of Matlab; execution is a lot faster than for either Octave or R. We can only conjecture that the performance gain is due to a more efficient language interpreter; casual comparison of other code examples would suggest that Matlab is consistently faster than Octave. However, a complete Matlab installation comes at a substantial price, especially a non-academic version. Those who are willing to acquire Matlab chiefly for its faster performance might want to consider buying a faster computer instead. These days, a rather decent (PC-based) workstation can be purchased for a lesser amount of money.

5 Extending Octave using C++

Extending Octave is quite straightforward. It requires that the operating system supports the creation of dynamically linked code using the GNU GCC compiler. This is surely the case for Linux, but not for Windows 95/98/NT. As an example, we discuss the implementation of a new random number

generator command via a wrapper around the 'Mersenne Twister' random number generator (RNG) by Matsumoto and Nishimura (1998). This generator uses a twisted generalized feedback shift-register algorithm which has a Mersenne prime period of $2^{19937} - 1 \approx 10^{6000}$ and is equi-distributed in 623 dimensions. It passes the 'DIEHARD' statistical tests by Marsaglia (1996) and at the same time is extremely fast and efficient in terms of memory usage. The wrapper code was written for an improved version by Shawn Cokus; this version is also available from the Mersenne Twister page at `http://www.math.keio.ac.jp/matumoto/emt.html`. The Mersenne Twister is a good example of how releasing a product of scholarly research under a well-understood license, the GNU GPL, leads not only to academic acclaim, but also to widespread actual implementations of the code in a variety of other open-source applications.

With the wrapper code in a file `randmt.cc` [3] and the actual RNG code in another file `cokus.c`, creating a compiled and dynamically loadable `.oct` file is as simple as typing `mkoctfile randmt.cc cokus.c`. This first compiles the random-number generator (from the file `cokus.c`) and then compiles and links the wrapper code (from the file `randmt.cc`) to form a new command `randmt.oct`. As any file with the extension `.oct` is a loadable module for Octave, it can be invoked from Octave as `randmt`. Generally, any valid C++ file can be translated with `mkoctfile` into a loadable module. C and Fortran can also be called from C++, but a discussion of this is beyond the scope of this review. If a command is implemented as both an `.m` and `.oct` file, the latter is given priority. It is also possible to call other Octave functions, whether implemented as `.oct` or `.m` files, which makes for a very powerful way to accelerate execution speed by re-writing time-critical components as `.oct` files. As mentioned above, the main problem with using C++ (or C or Fortran) to extend Octave is a lack of documentation about Octave's C++ API. On the other hand, tools like the open-source program `matwrap`, available at `http://lnc.usc.edu/~holt/matwrap/`, can aid in extending Octave. From a simple list of functions as presented in a header file, `matwrap` can generate all the required interface code for calling C++ code for Octave, or Matlab for that matter. This makes interfacing other C or C++ code much easier, and facilitates the integration with other libraries.

Just how much execution time can be saved by converting (interpreted) code from an `.m` file to (compiled) code in an `.oct` file? As a possible upper limit, we can consider the case of two nested loops around a very simple assignment, in this case the product of the current loop indices. This example is arguably pathological as it stresses the controlling outer loops over a very simple inner assignment. In real-life situation, more complex code would be placed inside the loops. Here, and for $n = 1000$, *i.e.* a matrix with a million elements, the interpreted `.m` file requires 108.54 seconds. But the compiled version requires only 0.18 seconds — this corresponds to a 600-fold increase in speed. While this example is arguably not a good representation of real-life applications, it still illustrates that the speed gain from compiled code can be very significant indeed.

6 Conclusion

This review has introduced the GNU Octave numerical programming environment by giving an extended overview, followed by a discussion of both positive and negative aspects of this language and environment. We have shown that data manipulation is straightforward with Octave. Using a simple Monte Carlo simulation, we provided an illustration of the performance of Octave both outright and in relation to R, the GNU implementation of the S language, and Matlab, a popular (but expensive)

[3] Due to space constraints, we have omitted the C++ code which is available at `http://rosebud.sps.queensu.ca/~edd/code/octave-mt.html` along with documentation and support files.

commercial closed-source package. Next, we have shown how the addition of a faster random number generator can improve performance quite significantly when Octave is used for simulation-based applications. Finally, a more general discussion of how to extend Octave with dynamically loadable code was provided, and illustrated with an (extreme case) of potential performance gains.

A key advantage of Octave is its GNU GPL license: copies of Octave can be given to colleagues, friends and students at will. The GNU R statistical environment is the only other major numerical program that is available under this license. While R, with its focus on modern statistics and data analysis, is interesting in its own right, Octave might be a more appropriate choice for primarily numerical work as the Octave language appears somewhat easier to learn, and already comprises a rich set of numerical functions and operators.

GNU Octave is a very powerful tool for doing numerical work, ranging from ad-hoc computations to elaborate simulation studies. Octave combines a clean and elegant language with a versatile interactive shell. This makes it very easy to prototype and implement new applications and procedures, both for applied work and new research. While Octave currently lacks more genuinely econometric applications, it has all the necessary ingredients for becoming a more common platform for econometricians.

References

Brown, Barry W., James Lovato, Kathy Russell, and John Venier (1997). RANDLIB: Library of `Fortran` routines for random number generation. Department of Biomathematics, M.D. Anderson Cancer Center, University of Texas, Houston. Source code is available at `ftp://odin.mdacc.tmc.edu/pub/source/`.

Cribari-Neto, Francisco (1997). Econometric programming environments: GAUSS, Ox and S-PLUS. *Journal of Applied Econometrics 12(1)*, 77–89.

Cribari-Neto, Francisco and Mark J. Jensen (1997). MATLAB as an econometric programming environment. *Journal of Applied Econometrics 12(6)*, 735–744.

Cribari-Neto, Francisco and Spyros G. Zarkos (1999). R: Yet another econometric programming environments. *Journal of Applied Econometrics 14(3)*, 319–29.

Eddelbüttel, Dirk (1996). Object-oriented econometrics: Matrix programming in C++ using GCC and Newmat. *Journal of Applied Econometrics 11(2)*, 199–209.

MacKinnon, James G. (1999). The Linux operating system: Debian GNU/Linux. *Journal of Applied Econometrics 14(4)*, 443–453.

Marsaglia, George (1996). DIEHARD: a battery of tests for random number generators. Available from the DIEHARD cdrom archive at `http://www.csis.hku.hk/internet/randomCD.html`.

Matsumoto, Makoto and Takuji Nishimura (1998). Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulation 8(1)*, 3–30. Code and information available at `http://www.math.keio.ac.jp/matumoto/emt.html`.

McCullough, Bruce D. (1999). Econometrics software reliablility: Eviews, LIMDEP, SHAZAM and TSP. *Journal of Applied Econometrics 14(2)*, 191–202.