JOURNAL OF APPLIED ECONOMETRICS
*J. Appl. Econ.* **14**: 319–329 (1999)

SOFTWARE REVIEWS

# R: YET ANOTHER ECONOMETRIC PROGRAMMING ENVIRONMENT

FRANCISCO CRIBARI-NETO[a]* AND SPYROS G. ZARKOS[b]

[a]*Dept. de Estatística, Universidade Federal de Pernambuco, Cidade Universitária, Recife/PE, 50740-540, Brazil*
[b]*Department of Economics, University of Athens, 8 Pesmazoglou str., Athens, Greece*

## SUMMARY

This article reviews R, an open-source S-like high-level matrix programming language that can be used for econometric simulations and data analysis. Copyright © 1999 John Wiley & Sons, Ltd.

## 1. INTRODUCTION

It used to be the case that someone doing applied econometrics could perform all required tasks using traditional econometrics software, such as SAS, SHAZAM, or TSP, to name only a few. Computer work was mainly a question of reading the manuals and identifying which of the pre-packaged routines could perform the desired task. Times have changed, however, and many newly developed techniques are not available in econometric packages. In order to use such techniques, one needs to *program* them. This has led to an explosion in the availability of 'econometric programming environments', that is, environments which provide users with a reasonably large number of econometric routines, but which also give users the flexibility to program new routines and modify existing ones. Such environments are a compromise between the greater flexibility offered by compiled languages such as C, C++, and FORTRAN and the convenience of traditional econometrics software. Well-known examples of econometric programming environments are GAUSS, Ox, and S-PLUS (Cribari-Neto, 1997) and MATLAB (Cribari-Neto and Jensen, 1997).[1] This paper reviews yet another econometric programming environment: R.

* Correspondence to: Francisco Cribari-Neto, Dept. de Estatística, Universidade Federal de Pernambuco, Cidade Universitária, Recife/PE, 50740-540, Brazil.

Contract/grant sponsor: CNPq.
Contract/grant sponsor: FINEP.

[1] For a quick tour of the S and S-PLUS environments, see Venables (1998). For a good guide to programming in S, see Chambers (1998). An older review of S-PLUS (for DOS and UNIX) is Hallman (1993).

CCC 0883-7252/99/030319–11$17·50
*Received 23 February 1999*

## 2. AN OVERVIEW

R was born when Ross Ihaka and Robert Gentleman, who were both interested in statistical computing and familiar with S, a programming language developed at the AT&T Bell Laboratories, became colleagues at the University of Auckland. Not pleased with the existing commercial software environment in their Macintosh teaching laboratory, Ihaka and Gentleman decided to make an effort to create their own programming environment. The initial goal was 'to demonstrate that it was possible to produce an S-like environment which did not suffer from the memory demands and performance problems which S has'. It was only later that they worked to turn R into a real and functional programming environment.

The belief that it was possible to create an improved (more efficient and functional) S-like environment through the incorporation of ideas borrowed from the Scheme programming language appears to have been the cornerstone of the early efforts. According to the authors (Gentleman and Ihaka, 1997; Ihaka, 1997, both unpublished manuscripts), the Scheme language (http://www.cs.indiana.edu/scheme-repository/home.html) was the basis of the experiment because of: (1) the availability of the source code of several Scheme interpreters, (2) its similarity to S, and (3) the availability of several books of reference, such as Abelson *et al.* (1985) and Kamin (1990). The initially developed interpreter (approximately one thousand lines of C code) is the backbone of the current version of R, providing a good deal of the language's functionality. When it was time to decide what type of user interface to create and how to implement the data structures necessary for the interpreter to be able to perform statistical work, Gentleman and Ihaka opted for an S-like syntax. 'This decision, more than anything else, has driven the direction that R development has taken' (Ihaka, 1997, unpublished manuscript).

In August 1993, the first version of R was announced on the S-news mailing list and became available to the public by means of a few copies of the software at StatLib (http://www.stat.cmu.edu/). Among the first people to show a pronounced interest in R was Martin Maechler of ETH Zürich. He joined Gentleman and Ihaka in the project and encouraged the release of the R source code under the terms of the Free Software Foundation's GNU Public License (June 1995).[2] From that point on, the development of R was no longer a closed process. With the strong interest expressed in the new language and significant contributions by a number of people, the modest initial project of Gentleman and Ihaka developed into a fully-fledged programming language. Since mid-1997, R has been jointly developed by an 'R core team'.[3]

A common definition of R is the following: 'R is an interpreted computer language designed for statistical data analysis.' More specifically, R is a programming environment which closely resembles the S programming language and its commercial enhanced version: S-PLUS (http://www.mathsoft.com/splus/). Indeed, R is often described as 'a statistical system not unlike S'. The S-like appearance though masks Scheme's heavy influence in the underlying implementation and semantics (mainly in the lexical scoping).

[2] In order to read the licensing terms, type ?license at the R prompt.

[3] The team consists of Doug Bates, Peter Dalgaard, Robert Gentleman, Kurt Hornik, Ross Ihaka, Friedrich Leisch, Thomas Lumley, Martin Maechler, Paul Murrell, Heiner Schwarte, and Luke Tierney. These people are academic statisticians and/or computer scientists, and R is their 'hobby' project.

R, unlike S and C, which by default use 'static' scoping, has implemented a 'lexical' scoping.[4] One way to observe the difference between the two rules is by comparing them at the level of the treatment of 'free variables'. In R (and also in Scheme), the value of a variable that is neither a formal parameter (i.e. a function argument) nor a local variable is determined by the bindings that were active at the time the variable was created. What matters here is the variable's defining environment. In S and C, values of free variables are determined by a set of global variables. For a detailed discussion of the implementation of the R language, see Gentleman and Ihaka (1998, unpublished manuscript), and Ihaka and Gentleman (1996).

The S and R syntaxes are so similar that the S 'blue book' (Becker *et al.*, 1988) can be used as an (approximate) R manual. An R manual entitled *Notes on R: A Programming Environment for Data Analysis and Graphics* by William Venables, David Smith, Robert Gentleman and Ross Ihaka is available from CRAN (The Comprehensive R Archive Network, a collection of sites carrying identical R material; see below for the WWW address). This manual resulted from a set of notes on S and S-PLUS written by the first two of the above-mentioned authors. As pointed out in the preface of the manual, it incorporates a number of rather modest changes to account for some small differences in the S and R syntaxes. R does come with a fairly detailed help in HTML format which can be accessed by typing `help.start()` at the R prompt. It causes a Web browser to open the main R help index page. Help on a particular R function in text mode can be obtained by typing a question mark followed by the function name or by typing `help` followed by the function name in parentheses at the R prompt, as for example `?read.table` or `help(read.table)`.

One of the main appealing features of R is undoubtedly the fact that it is free software, and hence can be obtained and distributed at no cost. Even the source code can be obtained from the R Web network (the CRAN master site is located in Vienna, Austria: `http://www.ci.tuwien.ac.at/R/`). This makes R not only a good programming environment for academic and professional econometricians, but also an excellent teaching tool; students can obtain the program at no cost, install it on their computers at home and learn econometrics through hands-on practice.

R is still in beta release, which means that there is still no official release of the environment. As mentioned earlier, the source code is available for download, and users can compile it under different operating systems. There are binaries available for Windows 95/98/NT, Linux, numerous versions of UNIX, and also for the Macintosh, although the latter is not as up-to-date as the R distributions for other platforms. Since R is written in ANSI C, installing it on Unix systems from the source code is in general automatic. This review was written using version 0.63.1 on a Pentium II 266 MHz with 128 MB RAM running on Windows NT Workstation 4.0.

Finally, since R is free software, it comes with no support. However, there are mailing lists which can be used to obtain expert advice from the members of the R core team and from experienced users. In order to subscribe to the `r-help` mailing list, one only needs to send an e-mail message to `r-help-request@stat.math.ethz.ch` with the word `subscribe` in the body (not the subject) of the message. It is a good idea to check the R frequently asked questions (FAQs) available at `http://www.stat.cmu.edu/R/CRAN/doc/FAQ/R-FAQ.html` before posting to the mailing list.

[4] Scoping refers to the rules by which variables, that is symbols and values, are associated. The interested reader is referred to Gentleman and Ihaka (1998, unpublished manuscript) and to Abelson *et al* (1995). Additionally, it should be noted that the concept is illustrated in R by means of an example if at the prompt one types `demo(scoping)`.

### 2.1. Matrix operations

One of the main advantages of a high-level programming language such as R is that users can program equations that require operations on matrices directly. That is, R comes with a set of pre-implemented routines to handle matrices. As an initial example, suppose we wish to generate a 50 × 2 matrix $X$ where the first column is a vector of ones and the second column consists of $\mathcal{U}(0, 1)$ random numbers:

```
X <- matrix(cbind(1,runif(50)), 50, 2)
```

('<-' is the assignment operator.) The sum of all elements of $(X'X)^{-1}$ can be easily obtained as

```
sum(solve(t(X)%*%X))
```

As with S-PLUS, t(X) yields the transpose of $X$, solve computes the inverse of a nonsingular matrix, and %*% is used for matrix multiplication, * being reserved for the Hadamard (direct) product of matrices. A function specifically tailored to efficiently invert symmetric matrices would be a nice addition to the language.[5]

R is a flexible programming environment, and as such it allows users to write their own functions and call them as if they were calling native R functions. For example, suppose we wish to write a function that when invoked computes the determinant of a square matrix $M$. This can be accomplished as

```
det <- function(M){
  Re(prod(eigen(M, only.values = T)$values))
}
```

After writing the det function, it can be invoked to compute, say, the determinant of $(X'X)^{-1}$ as

```
det(solve(t(X)%*%X))
```

which yields 0·006394678 as the result.

In order to print, say, the (13,2) element of $X$, all one has to do is to enter X[13,2] at the R prompt which results in 0·9028407. X[ ,2] prints the entire second column of $X$.

### 2.2. Mathematical operations

R comes with a wide range of built-in functions that can be used to perform many mathematical operations. Consider, for example, the polygamma functions. These are obtained by differentiating the loggamma function, $\log \Gamma(x)$. The first derivative of the loggamma function is the digamma function, denoted $\psi(x)$, whereas $\psi'(x)$ represents the trigamma function, $\psi''(x)$ denotes the tetragamma function, and so on. R includes built-in functions that evaluate the gamma, loggamma, digamma, trigamma, tetragamma, and pentagamma functions. For example, evaluating these functions at $x = 0{\cdot}5$ is an easy task in R:

```
c(gamma(0·5),lgamma(0·5),digamma(0·5),trigamma(0·5),tetragamma(0·5),
pentagamma(0·5))
[1]  1·772454  0·572365  −1·963510  4·934802  −16·828797  97·409091
```

[5] Both GAUSS and Ox have such a function.

the last line being the R output. The above polygamma functions are commonly encountered in econometric and statistical applications,[6] and R practitioners can numerically evaluate them with ease. It is noteworthy that only the first two functions (gamma and loggamma) are available in S-PLUS. Some of the other special functions available for use in R are: beta function (beta), log beta function (lbeta), binomial coefficients and their logarithms (choose and lchoose), trigonometric functions (cos, sin, tan, cosh, sinh, tanh, acosh, asinh, atanh), among others.

Pre-implemented functions to evaluate Bessel and modified Bessel functions are not available in R, as is the case in, say, GAUSS, MATLAB and Ox. However, because R is open-source software, it is more or less straightforward for any competent C programmer to implement these functions, or any others.

## 2.3. Graphics

Most high-level matrix programming languages are capable of producing publication quality graphics, and R is no exception. Like S and S-PLUS, R comes with a range of built-in functions for handling two- and three-dimensional plots. As an illustration, suppose we wish to generate a vector of values $y$ using the previously created $X$ matrix as $y = \beta_0 + \beta_1 x + \varepsilon$, where $x$ is the second column of $X$, $\varepsilon$ is a vector of independent random errors each distributed as $\mathcal{N}(0, 1)$, and $\beta_0$ and $\beta_1$ are taken to be equal to one. Next, we wish to plot $x$ against $y$, possibly as an initial visual inspection before regressing $y$ on $x$. This can be accomplished as

```
y <- X[ ,1] + X[ ,2] + rnorm(nrow(X))
plot(X[ ,2], y, xlab = "x", ylab = "y", main = "A plot of y versus x")
```

Three-dimensional graphics can be produced using the persp function whereas contour plots can be produced using the contour function. Future releases of R will include contour smoothing using B-splines for three-dimensional plots. Other useful graphics functions (whose names are self-explanatory) are barplot, boxplot, coplot, hist, image, pairs, piechart, qqnorm, qqplot. The best way to visualize the potential of R when it comes to producing a wide array of publication quality graphics is to type demo(graphics) at the R prompt and then navigate through a list of example plots.

The R functions for producing graphics are currently undergoing a major revision. Future releases are expected to have even better graphics capabilities.

## 2.4. Loops and other language constructs

Like any other programming language, R has language constructs for performing recurrent computations, such as loops. The loop syntax is quite simple. For example, for(i in 1:10) print(log(i)) is a simple loop which prints $\log(1), \log(2), \ldots, \log(10)$, where log denotes natural logarithm. Related constructs are while and repeat, with break being sometimes used within loops. It should be noted that S, S-PLUS, and R are notoriously slow when performing loops, although R seems to handle loops more efficiently than the others. Whenever possible, one should try to avoid loops by vectorizing the code. Some users may find this to be one of the main limitations of R.

[6] See e.g. Cordeiro and McCullagh (1991), Cordeiro *et al.* (1997), and Cribari-Neto and Ferrari (1995), among others.

R allows a function to call itself recursively. A good and well-known example is the factorial function, which does not come implemented into R but can be easily written as

```
factorial <- function(n){if (n <= 0) 1 else n * factorial(n - 1)}
```

After we write such a function, we can call it as we would call any built-in R function. For example, by entering factorial(10) at the prompt, we obtain the result 3628800.

### 2.5. Some differences between R and S

Although the R and S syntaxes are quite similar, there are a few minor differences which need to be pointed out. For example, in S the abs function, which computes the absolute value, works for both real and complex arguments, whereas in R it only works with reals. (In R, one should use the Mod function when working with complex numbers.)

Random number generation in R is more flexible than in S or in S-PLUS since R allows users to select from three different random number generators (RNGs), namely: 'Wichmann-Hill' (Wichmann and Hill, 1982), 'Marsaglia-Multicarry', and 'Super-Duper'.[7] The Marsaglia multiply-with-carry generator is one of the three RNGs implemented into Ox and has a period which exceeds $2^{60}$. It was posted to the usenet news group sci.stat.math by Professor Marsaglia on 29 September, 1997. Super-Duper consists of Marsaglia's traditional RNG and is currently used by both S and S-PLUS. Its period is $4{\cdot}6 \times 10^{18}$. A call to RNGkind suffices to choose among these three RNGs.

The Scheme-based lexical scoping of R implies a major difference in the way objects are stored: unlike in S, they are not saved as separate files but are kept internally (i.e., in memory). To that end, R dedicates a large amount of memory (specified by the user through the option --vsize) which it manages as efficiently as possible. There is a clear advantage associated with this approach: speed gains in execution time. The other side of the coin though is that unless one resorts to the strategy of saving images every so often (using the save.image function), should R crash one will lose all of the 'current-to-the-moment-of-the-crash' work. Also, while it is appealing to have functions that can maintain local state (i.e. preserve state information between function calls), the related disadvantage is that there is no straightforward (or simple) way of saving a function in R.

With respect to base level memory management, R is an environment that may allow a running program to use more memory than is physically present in the computer.[8] When a session starts, R occupies a portion of the computer's memory and manages it with a mark-sweep/compaction strategy known as 'garbage collection'. This process is set to motion when the Basic Language Elements (BLEs), where information is stored, in the memory heap are exhausted. First, the BLEs that are deemed necessary are marked, and, subsequently, the BLE array is swept and all unmarked array elements are put back to the list of available BLEs and moved in contiguous memory (compaction). This is where R and S/S-PLUS differ: the latter environments do not release memory properly and as a result, the amount of memory a running program uses can grow without a bound, thus slowing down the execution. The downside, however, of R's 'more

[7] There are currently plans to add even more random number generators in future releases of R.

[8] This is the case for programs that partition the executing image into memory 'pages', keep only a small subset of them in active memory and engage in a process of copying memory pages to and from the disk, as the need arises (process that is called 'paging'). With disk access being slow in comparison to memory access, performance is negatively related to paging and programs that need to page a lot are considerably slowed down.

reasonable' base level memory management is the following: if the amount of memory allocated at the start of the session is not sufficient for the running program to complete its execution, the job will be terminated at the point where all allocated memory is exhausted. As a result, the user will lose all the work performed until that point and will have to start all over, specifying what he or she believes to be enough memory for the program to execute.

Overall, it is safe to say that `R` is an 'implementation' of `S` since, with the exception of lexical scoping and its above-mentioned implications, the former follows the latter as closely as possible. Other occasional differences are intentional and aim at improving the language's clarity and at facilitating debugging.

### 2.6. Packages for `R`

The standard distribution of `R` includes the following libraries: `base` (the `R` base package of functions), `eda` (exploratory data analysis), `modreg` (modern regression: smoothing and local methods), `mva` (multivariate analysis), and `stepfun` (step functions, including empirical distributions). 'Eda' contains functions for robust fitting, median polish and smoothing whereas 'mva' provides code for principal components, canonical correlations, hierarchical clustering and metric multidimensional scaling analysis, among other things.

Many more add-on packages exist for `R` at the CRAN contributed code area. `MASS` is the main package from Venables and Ripley (1997). The extension package boot contains code and datasets from Davison and Hinkley (1997), and `bootstrap` is the package of functions that goes along with Efron and Tibshirani (1993). There are also `R` packages for kernel smoothing and density estimation (`KernSmooth`), for logspline density estimation (`logspline`), for spline regression (`splines`), for quantile regression and related rank statistics (`quantreg`), for solving quadratic programming problems (`quadprog`), for maximum likelihood estimation of fractionally integrated processes (`fracdiff`), for dynamic system estimation (`dse`), for cluster analysis (`cluster`), and for survival analysis (`survival4`), to name only a few of the additional add-ons to `R` available from `CRAN`.[9]

It is noteworthy that some of these add-on libraries can be quite useful in supplementing and extending the `R` capabilities. A prime example is the case of 'mean absolute deviation' (MAD) regression. Unlike `S-PLUS`, `R` does not come with an `l1fit` function for estimating $l_1$ (i.e. MAD) regression models. However, this can be accomplished in `R` by using the `quantreg` library, since $l_1$ regression is a special case of quantile regression. The `quantreg` library has been developed by Roger Koenker and was ported to `R` by Kjetil Halvorsen. For a detailed account of algorithms for quantile regression estimation, see Portnoy and Koenker (1997).

## 3. A WEB INTERFACE TO `R`

The `Rweb` is a Web interface to `R` and is available at `http://www.math.montana.edu/Rweb/`. Users can run `R` programs from the `Rweb` home page in batch mode and obtain the output from the program (which may include graphics) directly in their Web browsers. Basically, the user types in the code, clicks on a 'submit' button, and a page with the results (analysis and graphics) is returned. The source code and other relevant information for those who would like to set up their own `Rweb` page can be obtained from the `Rweb` resources home page, currently at `http://`

[9] For a complete list, see `http://www.stat.cmu.edu/R/CRAN/src/contrib/PACKAGES.html`.

www.math.montana.edu/Rweb/Resources.html. The Rweb interface has been set up by Jeff Banfield, an Associate Professor of Statistics at Montana State University.

## 4. A SIMPLE MONTE CARLO EXAMPLE

Suppose we wish to write a simple Monte Carlo simulation program that simulates a normal linear regression model, for teaching purposes. In particular, suppose we wish to replicate the simulation results on pages 219–223 of Griffiths *et al.* (1993). The model under study is a simple normal linear regression model: $y_t = \beta_1 + \beta_2 x_t + e_t$, where $e_t \sim \mathcal{N}(0, \sigma^2)$, the $e_t$'s being independent. The goal is to use the ordinary least squares estimates $b_1 = 7{\cdot}3832$, $b_2 = 0{\cdot}2323$ and $\hat{\sigma}^2 = 46{\cdot}853$ (given on page 219) as the true parameter values and then perform a simulation experiment. At the end of the simulations, we will produce a histogram of the different values of $b_2$, thus replicating the figure on page 222 of the book. This is a nice introductory exercise since it teaches students how to perform a simple Monte Carlo simulation. The R function given below performs such a task.

```
MC.sim <- function (r = 1000){
  RNGkind(kind = "Super-Duper")
  if (r <= 0)
    stop("The number of replications (r) must be positive!")
  beta1 <- 7·3832; beta2 <- 0·2323; sigma2 <- 46·852
  x <- c(25·83, 34·31, 42·5, 46·75, 48·29, 48·77, 49·65, 51·94,
    54·33, 54·87, 56·46, 58·83, 59·13, 60·73, 61·12, 63·1,
    65·96, 66·4, 70·42, 70·48, 71·98, 72, 72·23, 72·23, 73·44,
    74·25, 74·77, 76·33, 81·02, 81·85, 82·56, 83·33, 83·4,
    91·81, 91·81, 92·96, 95·17, 101·4, 114·13, 115·46)
  T <- length(x); x <- as.matrix(x); X <- cbind(1, x)
  y.simulated <- beta1 + beta2 * X[, 2] + matrix(rnorm(T *
    r, mean = 0, sd = sqrt(sigma2)), T, r)
  estimate <- solve(t(X) %*% X) %*% t(X) %*% y.simulated
  hist(estimate[2, ], col = "gray", breaks = 12, xlab = "b2",
    main = "histogram of b2", ylab = "frequency", xlim = c(0, 0·5))
  cat("\nMean estimate: ")
  return(mean(estimate))
}
```

The function can now be invoked from the R prompt as MC.sim() (it will use the default number of replications, 1000). The output of the call is a histogram, which appears in the graphics window, and the mean of all estimates $b_2$, which is displayed in the commands window. The histogram produced by R is given in Figure 1. Note that the above Monte Carlo program does not make use of any loops. Instead, the code has been vectorized in order to run more efficiently. It is possible to write a similar program that is loop-based by replacing the line that starts with 'estimate <-' by

```
M <- solve(t(X) %*% X) %*% t(X)
estimate <- matrix(0, 2, r)
for(i in 1:r){
  estimate[, i] <- M %*% y.simulated[, i]
  }
```

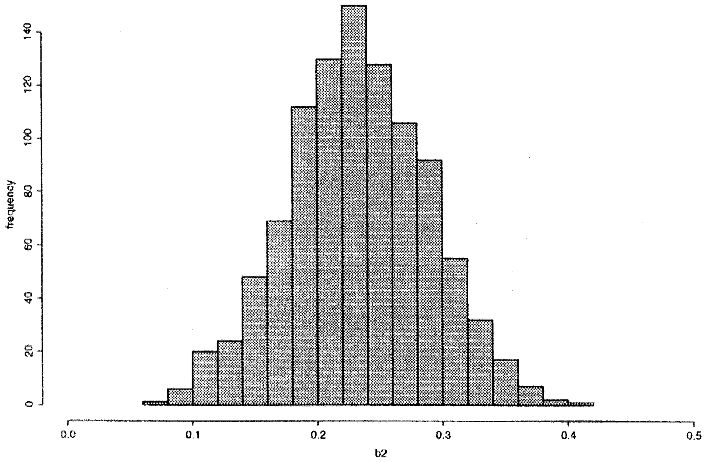

Figure 1. A histogram

Table I. Execution times for the OLS Monte Carlo program

| *r* | R (vec) | R (loop) | S-PLUS (vec) | S-PLUS (loop) |
|---|---|---|---|---|
| 1000 | 0·56 | 0·59 | 0·39 | 3·92 |
| 5000 | 1·26 | 1·60 | 0·81 | 18·50 |
| 10000 | 2·37 | 2·92 | 1·35 | 36·64 |
| 50000 | 10·64 | 14·03 | 5·79 | 205·98 |

## 5. SPEED EVALUATION

We used the simulation program in the previous section to assess the efficiency of R. It was run on R version 0.63.1 for Windows and on S-PLUS 4.5 for Windows for $r = 1000, 5000, 10000, 50000$. The timings (in seconds) for R and S-PLUS for both the vectorized and the loop-based program are given in Table I. All times are the smallest execution time from three consecutive runs.[10] R and S-PLUS are roughly equally efficient when the code is vectorized, S-PLUS being a bit faster. However, when the program is based on a simulation loop where $r$ estimations are performed sequentially, R becomes much faster than S-PLUS. For example, when we set the number of replications to 50,000, R is nearly 15 times faster than S-PLUS.

The above example suggests that R is more efficient than S-PLUS in handling loops. In order to investigate this further, consider an example drawn from Cribari-Neto and Jensen (1997) which consists of using a double loop to create a matrix $M$ of dimension $n \times n$ whose $(i, j)$ entry equals $i + j$. Of course, double looping is not an efficient way of constructing $M$. Our purpose is simply to use this example as a measure of loop speed. We have modified Cribari-Neto and Jensen's

[10] The program was slightly altered to run under S-PLUS

(1997) code by adding the computation of the sum of all elements of $M$. The R (and S-PLUS) code is:

```
loop.matrix <- function(n){
  M <- matrix(0, n, n)
  for(i in 1:n){
    for( j in 1:n){
      M[i, j] <- i + j
    }
      }
  return(sum(M))
}
```

The execution times for different values of $n$ are given in Table II. All entries are seconds and (again) represent the best of three consecutive timings.

Once more, R is faster than S-PLUS, more so as the loop length gets larger. For example, when the dimension of $M$ is 500 × 500, R is over twice as fast as S-PLUS. Even though R runs faster than S-PLUS when handling loops, it can still be substantially slower than other econometric environments. As a frame of reference, we have run this program using Ox for Windows version 2.0a on the same hardware. The execution times for $n = 100, \ldots, 500$ were 0·05, 0·20, 0·45, 0·80, 1·17 second, respectively. If we take $n = 500$ for example, we see that Ox is 22 times faster than R (and nearly 47 times faster than S-PLUS).

Of course, users may want to consider low-level programming languages such as C (Cribari-Neto, 1999), C++ (Eddelbüttel, 1996) and FORTRAN for tasks which are very computer-intensive, such as, for example, the simulation of a double bootstrapping scheme. An alternative strategy is to code the most computer intensive part of the program in C and link the compiled C code to R.

## 6. CONCLUDING THOUGHTS

There is an increasing demand for econometric programming environments which combine the ease of use of traditional statistical software with the flexibility provided by a programming language. S-PLUS has been the primary choice in the statistics community and has been gaining some ground among econometricians recently. R is an appealing alternative to S-PLUS for two reasons. First, it is free software. Second, it tends to be more efficient in terms of speed and memory usage than S-PLUS, benefiting from the use of a different 'underlying engine', and yet with nearly the same syntax. At the moment, R is still under being beta tested, but we believe it has the potential to become a useful tool for data analysis, programming, and teaching in the econometrics community.

Table II. Execution times for a double loop program

| $n$ | R | S-PLUS |
|---|---|---|
| 100 | 1·03 | 1·06 |
| 200 | 4·11 | 4·38 |
| 300 | 9·20 | 15·58 |
| 400 | 16·29 | 32·63 |
| 500 | 25·74 | 54·77 |

## ACKNOWLEDGEMENTS

We wish to thank Jurgen Doornik, Robert Gentleman, Roger Koenker, James MacKinnon, and Álvaro Novo for comments on earlier drafts. The usual disclaimer applies. The first author gratefully acknowledges financial support from the Brazilian agencies CNPq and FINEP.

## REFERENCES

Abelson, H., G. J. Sussman and J. Sussman (1985), *Structure and Interpretation of Computer Programs*, MIT Press, Cambridge.

Becker, R. A., J. M. Chambers and A. R. Wilks (1988), *The (New) S Language: A Programming Environment for Data Analysis and Graphics*, Wadsworth & Brooks/Cole, Pacific Grove.

Chambers, J. M. (1998), *Programming with Data: A Guide to the S Language*, Springer-Verlag, New York.

Cordeiro, G. M. and P. McCullagh (1991), 'Bias correction in generalized linear models', *Journal of the Royal Statistical Society B*, **53**, 629–643.

Cordeiro, G. M., E. C. Rocha, J. G. C. Rocha and F. Cribari-Neto (1997), 'Bias-corrected maximum likelihood estimation for the beta distribution', *Journal of Statistical Computation and Simulation*, **58**, 21–35.

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

Cribari-Neto, F. (1999), 'C for econometricians', *Computational Economics*, forthcoming.

Cribari-Neto, F. and S. L. P. Ferrari (1995), 'Second order asymptotics for score tests in generalised linear models', *Biometrika*, **82**, 426–432.

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

Davison, A. C. and D. V. Hinkley (1997), *Bootstrap Methods and Their Application*, Cambridge University Press, New York.

Eddelbüttel, D. (1996), 'Object-oriented econometrics: matrix programming in C++ using GCC and newmat', *Journal of Applied Econometrics*, **11**, 199–209.

Efron, B. and R. J. Tibshirani (1993), *An Introduction to the Bootstrap*, Chapman and Hall, New York.

Gentleman, R. and R. Ihaka (1997), 'The R language', In L. Billard and N. Fisher (eds), *Proceedings of the 28th Symposium on the Interface*, The Interface Foundation of North America.

Griffiths, W. E., R. C. Hill and G. G. Judge (1993), *Learning and Practicing Econometrics*, Wiley, New York.

Hallman, J. (1993), 'Review of S-PLUS', *Journal of Applied Econometrics*, **8**, 213–219.

Ihaka, R. and R. Gentleman (1996), 'R: a language for data analysis and graphics', *Journal of Computational and Graphical Statistics*, **5**, 299–314.

Kamin, S. N. (1990), *Programming Languages*, Addison-Wesley, New York.

Portnoy, S. and R. Koenker (1997), 'The Gaussian hare and the Laplacian tortoise: computability of squared-error versus absolute-error estimators', *Statistical Science*, **12**, 279–300.

Venables, W. (1998), 'S and S-PLUS: an eclectic tour of the environment', *Computational Statistics*, **13**, 27–46.

Venables, W. N. and B. D. Ripley (1997), *Modern Applied Statistics with S-PLUS*, Springer-Verlag, New York.

Wichmann, B. A. and I. D. Hill (1982), 'Algorithm AS183: an efficient and portable pseudo-random number generator', *Applied Statistics*, **33**, 123.