LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

INSTITUT FÜR STATISTIK

Friedrich Leisch

R behind the scenes: Using S the (un)usual way

Technical Report Number 012, 2007
Department of Statistics
University of Munich

http://www.stat.uni-muenchen.de

R behind the scenes: Using S the (un)usual way

Friedrich Leisch
University of Munich, Department of Statistics
Ludwigstrasse 33
80539 Munich, Germany
E-mail: Friedrich.Leisch@stat.uni-muenchen.de

This is a preprint of an article published in:
Proceedings of the 56th Session of the International Statistical Institute, IPM36, Lisbon, Portugal, 2007.

1 Introduction

Most users know R (R Development Core Team, 2007a) as a statistical computing environment presenting them a prompt or minimalistic GUI for data analysis. The user enters data and commands, and R responds with figures, tables, fitted models, etc.. A graphical representation of this beginner's view of R is shown on the left side of Figure 1. However, as every novice realizes soon (and sometimes not without pain), behind the prompt R is first of all an interpreter for a programming language named S. The origin of the language was the wish for interactive access to a set of Fortran data analysis subroutines at Bell Labs (Becker, 1994), which gradually evolved into the full-featured object-oriented language we know today as S version 4 (Chambers, 1996). *"Nothing is more important for the success of statistical software than enabling the transition from user to programmer, and on to gradually more ambitious software design"* (Chambers, 2000).

As a user makes progress in mastering R, he or she will soon realize that R cannot only read data, but also write data, and that behind the S language there is code written in Fortran or C to do the number crunching, visualized on the middle in Figure 1. Data written out is not limited to mere copies of the date previously read in: R possesses a number of string processing capabilities like regular expression handling, it can aggregate and reshape data, perform join operations on multiple data sets, and much more. Section 3 shows an application of using R as a text processor.

Another important aspect of the S language is that its origin – a set of macros that could be used as a glue for independent software routines written in programming languages like C or Fortran – is still a very good reason to use it. Many bits and pieces of R are written in compiled languages. One reason is speed, number crunching in C can be orders of magnitude faster than number crunching in S. Another important reason is that R tries not to reinvent the wheel wherever possible. The statistical community has developed a huge body of numerical routines over the last decades, ranging from distribution functions and random number generators to parameter estimation for complicated models.

R provides access to these routines within a unified environment. The base distri-

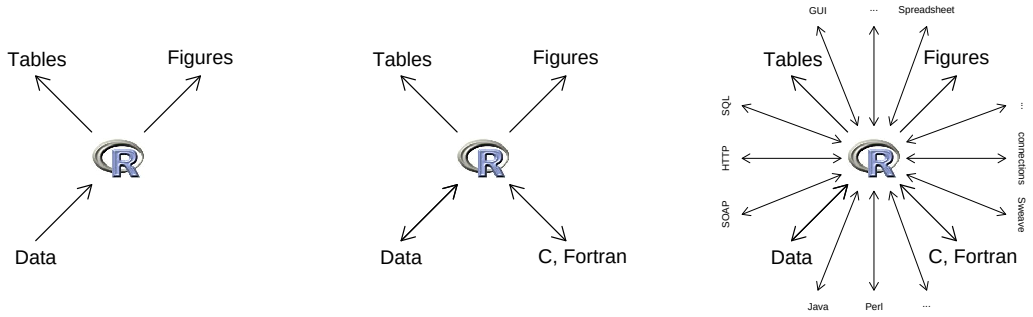

Figure 1: A beginner's view of R (left), a beginning developer's view of R (middle), and a full view of R (right).

bution ships many numerical routines which are copies (or based on copies of) well-tested and published algorithms. Of course, permission by the respective copyright owners is important here, see the file `doc/COPYRIGHTS` in the R sources for details. To give just one example, quantiles of the normal distribution are based on algorithms AS 111 and AS 241 (Beasley and Springer, 1977; Wichura, 1988). Because R can be easily extended using R packages (R Development Core Team, 2007b), it nowadays forms not only the glue between a set of independent software routines. It acts as a broker of methodology in computational statistics, making both "old classics" and "new state-of-the-art" available to everyone at a single prompt and within a single programming language.

The remainder of this article will give a short overview of how to use this huge collection of statistical methodology in other ways than entering commands at the R prompt. The prompt is only one way of utilizing R, and numerous other ways have been developed over the last years: embedding R in other applications like spreadsheets, dynamic statistical documents combining text and code, using R as a scripting language, or as a webpage plugin offering a wide range of services from simple examples for teaching to complete data analyses over the Internet. Of course only the tip of the iceberg can be shown in 8 pages, and this article is not meant as a comprehensive survey. Rather than listing all known applications, it will concentrate on a few examples and cover those in more detail, including full code listings where possible.

2 Reasons for Embedding R

There are numerous reasons for using R "behind the scenes" of another application, but the main motivation behind most is one or both of the following two:

1. Controlling R in other ways than entering commands at the prompt.

2. Directly use the results of R computations with another program.

The R console connects R's input to the keyboard and output to the screen, by embedding R we can use a program rather than a human at either end. If R produces a

table of numerical results and that table is meant to be part of a manuscript, then the computer should insert the table into the manuscript. Manual copying costs time and always has the risk of making a mistake. If a standardized analysis allows only for a limited number of operations at a certain point, it may be more convenient to choose among menu items in a graphical user interface rather than entering a command at the prompt. The full power of the S language is convenient in many places, but limiting the number of choices can reduce the probability of human error.

The following three examples from the author's own practical work may give an idea when using R at the prompt is not ideal:

Report Generation: A data analysis is finished, numerous tables and figures have been created and form an integral part of the report. Now if the data change slightly (e.g., a typing error in the original data is detected), then we do not need a new interactive R session to redo the complete analysis. Recalculating an analysis and inserting the new results at the right places can be fully automated such that humans only have to check whether the conclusions drawn are still valid or need to be updated.

Routine Analysis: In a cooperation project with a software company we developed a statistical model for direct marketing actions. Based on purchase data, affinities of consumers to product groups are identified and used by a recommender system. The data warehouse of the client is updated every week with new data, R then automatically recalculates the affinities of all registered costumers to all product groups, stores the affinities back into the data warehouse, and sends an email with a summary sheet to the marketing department. No human interaction is needed unless the summary report indicates a problem.

Agent-based Simulations: As part of a research center of excellence on "adaptive information systems in management science" our research group was conducting large scale simulations on workstation clusters. Several R processes where running simultaneously on different computers, each representing an agent in an artificial economy (e.g., different companies trying to sell similar products to virtual customers). In each iteration of the simulation, each agent analyzed the available market data and positioned itself in the a seemingly profitable market segment. The goal of the simulation was to see whether known stylized facts of real world markets can be reproduced with simple (but non-trivial) artificial agents. Because agents needed to be able to perform statistical analyses, using R to implement the simulation was a natural choice. Connecting several R processes to a large simulation allowed to efficiently use a cluster of workstations and also have some agents written in Octave or MATLAB (Meyer et al., 2003).

3 Embedding R in Text Documents

As mentioned above, S is not merely a language for statistical data analysis, it is a full-featured programming language designed for interactive use. The R implementation

is under the hood a Scheme interpreter (Ihaka and Gentleman, 1996), i.e., R belongs to the Lisp family of programming languages, although this is not apparent from the S syntax used. Over time, we have added many functionalities to R which are not directly needed for analyzing or visualizing data in a strict sense. One can download files over the Internet (install packages from CRAN, ...), has access to the operating system (create, list, copy and delete files, ...), and much more. It takes only a few lines of code in R to send serial emails, AKA spam.

Probably one of the most underused (in terms of user numbers) features of R is using regular expressions for string processing, see `help("regexp")`. A simple version of regular expressions are wildcards for listing files, the command `ls *.tex` will list all files with extension `.tex` on a Unix system, because these files match the expression `*.tex`. Regular expression adhere to the same principle, but have a much richer syntax and allow to do very complicated match and replace operations. The user can search for certain patterns in strings and replace them by other patterns. Several R functions use regular expressions, `grep()` (find patterns), `sub()` (find&replace patterns) and `help.search()` are perhaps the most prominent ones. Following the principle of not reinventing the wheel, R does not have its own implementation of regular expressions, but uses the GNU and PCRE regexp libraries.

One important application in statistics is for pre-processing data. But regular expressions can also be used to do text processing in R. Sweave (Leisch, 2002) allows to embed R code directly into latex documents. When Sweave processes such a document, it identifies the R code, evaluates it, and inserts the resulting output (text, figures) into the document. Figure 2 shows the code used to generate Figure 1. The regular expression R uses to identify where a "code chunk" (piece of R code to evaluate) starts is `^<<(.*)>>=.*`. The `^` at the beginning means that the pattern must start in column 1, then there must be exactly two "less than" signs, followed by an arbitrary sequence, followed by two "greater than" and one "equal" sign.

None of this is directly related to statistical data analysis, but it means that R can do text processing, and hence can easily be used for reproducible research where analysis code is tightly linked to reports describing the analysis (Leisch and Rossini, 2003). Packages `R2HTML` (Lecoutre, 2003) and `odfWeave` (Kuhn, 2006) on CRAN provide adaptations of Sweave which allow to use HTML or OpenOffice for word processing rather than latex. In addition, Figure 1 shows that R is not only useful for creating statistical graphs like scatterplots or boxplots, but can be used as a programmable drawing program, see Murrell (2005) for more examples, which are also available as package `RGraphics` on CRAN.

Sweave and friends embed R code into text documents, not R itself. The code can be dynamically replaced by the output of evaluating the code, but the document itself is static. In many cases this is the right thing to do, but sometimes a more interactive version is more appropriate. Package `Rpad` (Short and Grosjean, 2006) is one of several implementations that embed R directly into a webpage: the browser connects to a running R process, commands are entered through HTML forms as free text or through elements like pulldown menus and radio buttons.

```
<<>>=
library(pixmap)
logo <- read.pnm(system.file("pictures/logo.ppm",
                                        package="pixmap"))
@

<<diag1,fig=TRUE>>=
par(mar=rep(0,4))
plot.new()
plot.window(xlim=c(0,1),ylim=c(0,1))
addlogo(logo, c(0.42, 0.58), c(0.44, 0.56))
text(0.25, 0.8, "Tables", cex=2)
text(0.75, 0.8, "Figures", cex=2)
text(0.25, 0.2, "Data", cex=2)
arrows(0.25, 0.25, 0.42, 0.42, lwd=2, code=2)
arrows(0.25, 0.75, 0.42, 0.58, lwd=2, code=1)
arrows(0.75, 0.75, 0.58, 0.58, lwd=2, code=1)
@

<<diag2,fig=TRUE>>=
<<diag1>>
text(0.8, 0.2, "C, Fortran", cex=2)
arrows(0.25, 0.25, 0.42, 0.42, lwd=2, code=3)
arrows(0.75, 0.25, 0.58, 0.42, lwd=2, code=3)
```

Figure 2: Source code for Figure 1 as Sweave file.

```
#include <Rembedded.h>

int main(int ac, char **av)
{
    Rf_initialize_R(ac, av);
    Rf_mainloop(); /* does not return */
    return 0;
}
```

Figure 3: Minimal C code to run the R interpreter.

4 Embedding R in Programs

Rpad is one example where the user communicates with R not directly via the R prompt or by executing R scripts, but through an alternate frontend: in this case a webpage. Embedding R into other programs is much simpler than many users or developers think, and has been so for quite some time now (e.g., Temple Lang, 2001).

The most common usage for embedding R is to write alternative graphical user interfaces to the interpreter, see `http://www.r-project.org/GUI`. R itself is written in C, it can be built as a shared library and linked into any other application that can access C code. Figure 3 shows the minimal C code necessary: compiling the code listed and linking it against `libR` will result in an executable that starts the R engine and presents the user a fully functional R prompt (if the right environment variables are set, see the *Writing R Extensions* manual). In fact, the terminal version of R does in essence the same thing, plus some additional setup and error handling. Of course it makes only limited sense to replicate how one creates a terminal version of R. But the example demonstrates that only few lines of code are necessary, e.g., to add a window running an R process to another application.

Directory `tests/Embedding` of the R sources contains several examples of embedding R in C, including how to directly communicate from C code with the R interpreter (rather than starting a prompt waiting for user input). Communication with R works not only for C programs, R has interfaces to many other programming languages, most of which have been developed as part of the Omegahat project (Chambers and Temple Lang, 2001). Urbanek (2007) discusses some more recent developments.

Linking R into an application is one way of embedding R, the other main way is to talk to a running R process using a communication protocol. This has the advantage that R and the embedding program need not necessarily run on the same computer, but can also be started on different machines. Again, the basic ingredients are surprisingly simple. Many R functions that can read or write from/to files accept also so-called connections (Ripley, 2001) instead of a file on the local disk.

One connection type are sockets, which allow R to communicate with other programs over a network. Figure 4 shows R code for a simple R server: function `simpleServer()` first opens a connection on a user-specified port. Because we use `server=TRUE`, R will wait for other processes to connect to it. When it receives a call, it

```
simpleServer <- function(port=6543)
{
    sock <- socketConnection(port=port, server=TRUE)
    on.exit(close(sock))
    cat("\nWelcome to R!\nR> ", file=sock)

    while((line <- readLines(sock, n=1)) != "quit")
    {
        cat(paste("socket>", line, "\n"))
        out <- capture.output(try(eval(parse(text=line))))
        writeLines(out, con=sock)
        cat("\nR> ", file=sock)
    }
}
```

```
shell> telnet localhost 6543
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

Welcome to R!
R> summary(iris[,3:5])
  Petal.Length    Petal.Width          Species
 Min.   :1.000   Min.   :0.100   setosa     :50
 1st Qu.:1.600   1st Qu.:0.300   versicolor:50
 Median :4.350   Median :1.300   virginica :50
 Mean   :3.758   Mean   :1.199
 3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :6.900   Max.   :2.500

R> quit
Connection closed by foreign host.
```

Figure 4: Talking to R via sockets: An R function implementing a minimalistic server (top) and transcript of communication with the server (bottom).

first responds by returning a welcome message, then it enters an infinite loop. Every line it receives will first be tested if it equals the special keyword `"quit"`, in which case the connection is closed and the function returns. All other text that is received is assumed to be R commands, and `simpleServer()` tries to evaluate it. Textual results are sent back to the client.

The lower panel of Figure 4 shows how a client can connect to our simple server. Any program that can communicate over a socket could be used for this purpose, including R itself. The example uses the `telnet` command line utility, which is available for all major operating systems. First we need to start the server (not shown in Figure 4) by executing `simpleServer()` at the prompt of an R process. After that, the client can connect. If the client runs on the same machine, entering `telnet localhost 6543` will connect to our server. If the client runs on a different machine, then `localhost` needs to be replaced by the IP address of the machine running the R server. The port number 6543 is arbitrary, any port not already used by another program could be used.

The server shown in the example is of course only good for demo purposes, because no client authentification whatsoever is done. Anybody who guesses machine and port correctly can connect, and there are programs doing automatic port scans. So we should at least ask the client for a password or use some other means of authentification/protection. Behind a firewall or in an intranet socket communication without any authentification may however be a simple and sufficient solution.

Package `Rserve` (Urbanek, 2006) provides more sophisticated support for connecting to a running R process over a socket. It allows for multiple simultaneous connections (with seperated workspaces), authentification and transparent transfer of complete R objects. Client-side implementations are available for C, C++, Java and R. E.g., the following code from the Rserve homepage at `http://rosuda.org` connects a Java program to R and gets 10 Gaussian random variables from R into a Java array:

```
Rconnection c = new Rconnection();
double d[]=c.eval("rnorm(10)").asDoubleArray();
```

This now connects the full circle from the beginning: We go from Java to R, R itself interfaces several state-of-the-art random number generators implemented in C and FORTRAN. Of course we could directly link the Java application to the random number generator, and if only these 10 numbers are needed, that is certainly the way to go. But if more statistical methods are needed, using R as an intermediate layer may be the easier way.

Sockets are only one form of inter-process communication over the Internet, several others exist. DCOM is a communication protocol for connecting applications on Microsoft Windows systems (again either on the same or different machines). The R-Excel interface (Baier and Neuwirth, 2007) embeds R into the Excel spreadsheet, see Figure 5. R knowledge is only necessary to write sheets using R, but the pre-fabricated sheets can then be passed on to other users who have no knowledge of R. Data are entered directly into Excel, for the naive user there is no visible difference if R or Excel calculates the results.

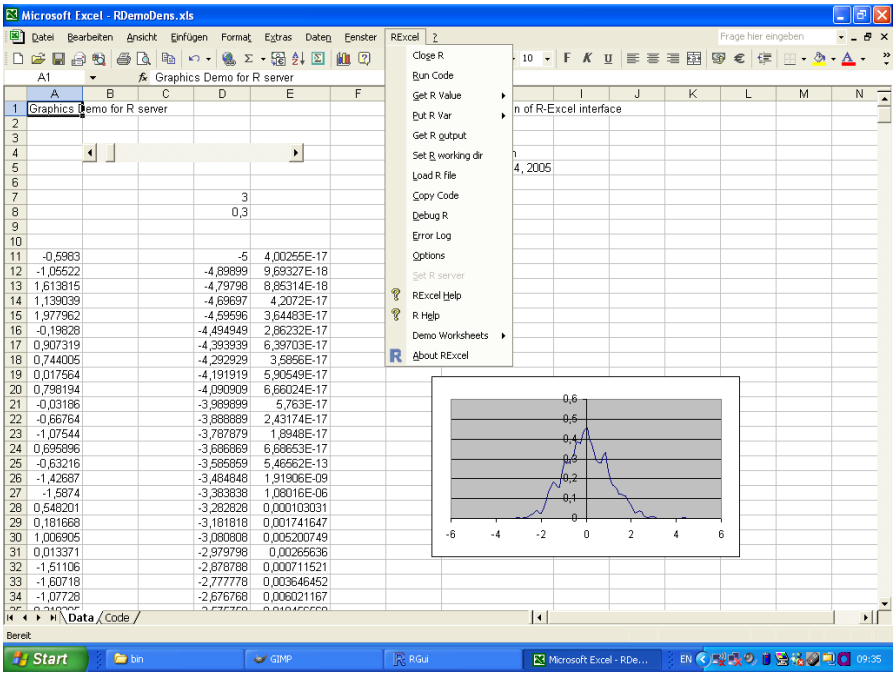

Figure 5: Screenshot of the R-Excel interface.

Acknowledgements

R is a is the result of a collaborative effort and much of the software presented in this paper has been implemented by other members of the R Development Core Team than the author of this article, or by members of the very active R developer community, without whom R would not be what it is today.

References

Baier, T. and Neuwirth, E. (2007). Excel :: Com :: R. *Computational Statistics*, 22:91–108.

Beasley, J. D. and Springer, S. G. (1977). Algorithm AS 111: The percentage points of the normal distribution. *Applied Statistics*, 26:118–121.

Becker, R. A. (1994). A brief history of S. In Dirschedl, P. and Ostermann, R., editors, *Computational Statistics – Papers Collected on the Occasion of the 25th Conference on Statistical Computing at Schlosz Reisensburg*, pages 81–110. Physica, Heidelberg, Germany.

Chambers, J. M. (1996). Evolution of the S language. In *20th Symposium on the Interface*.

Chambers, J. M. (2000). Users, programmers, and statistical software. *Journal of Computational and Graphical Statistics*, 9(3):404–422.

Chambers, J. M. and Temple Lang, D. (2001). Omegahat packages for R. *R News*, 1(1):21–24.

Ihaka, R. and Gentleman, R. (1996). R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314.

Kuhn, M. (2006). Sweave and the open document format – the odfWeave package. *R News*, 6(4):2–8.

Lecoutre, E. (2003). The R2HTML package. *R News*, 3(3):33–36.

Leisch, F. (2002). Sweave: Dynamic generation of statistical reports using literate data analysis. In Härdle, W. and Rönz, B., editors, *Compstat 2002 — Proceedings in Computational Statistics*, pages 575–580. Physica Verlag, Heidelberg. ISBN 3-7908-1517-9.

Leisch, F. and Rossini, A. J. (2003). Reproducible statistical research. *Chance*, 16(2):46–50.

Meyer, D., Buchta, C., Karatzoglou, A., Leisch, F., and Hornik, K. (2003). A simulation framework for heterogeneous agents. *Computational Economics*, 22(2):285–301.

Murrell, P. (2005). *R Graphics*. Chapman & Hall / CRC, Boca Raton, USA.

R Development Core Team (2007a). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.

R Development Core Team (2007b). *Writing R Extensions*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-11-9.

Ripley, B. D. (2001). Connections. *R News*, 1(1):16–17.

Short, T. and Grosjean, P. (2006). *Rpad: Workbook-style, web-based interface to R*. R package version 1.2.1.

Temple Lang, D. (2001). Embedding S in other languages and environments. In Hornik, K. and Leisch, F., editors, *Proceedings of the 2nd International Workshop on Distributed Statistical Computing, March 15-17, 2001, Technische Universität Wien, Vienna, Austria*. ISSN 1609-395X.

Urbanek, S. (2006). *Rserve: Binary R Server*. R package version 0.4-7.

Urbanek, S. (2007). How to talk to strangers: Ways to leverage connectivity between R, Java, and Objective C. Unpublished manuscript submitted to proceedings of DSC 2007, AT&T Labs Research, USA.

Wichura, M. J. (1988). Algorithm AS 241: The percentage points of the normal distribution. *Applied Statistics*, 37:477–484.

RESUME

R is not only a program for analyzing and visualizing data, it is an open and programmable software environment. It can not only easily access other programs written in a wide variety of languages, but also be accessed itself from other programs. As such, it can be seen as the computational Swiss army knife of statistics. Connecting a program to R can be surprisingly simple, and once the connection is established, the perhaps largest existing collection of statistical methodology is available through a unified interface. Embedding R can save a lot of human time by automating routine tasks, but more importantly, it often gives a simple way of making our methods accessible to a much wider audience.