R News

The Newsletter of the R Project Volume 1/1, January 2001

Editorial

by Kurt Hornik and Friedrich Leisch

Welcome to the first volume of *R News*, the newsletter of the R project for statistical computing. R News will feature short to medium length articles covering topics that might be of interest to users or developers of R, including

- Changes in R: new features of the latest release
- Changes on CRAN: new add-on packages, manuals, binary distributions, mirrors, ...
- Add-on packages: short introductions to or reviews of R extension packages
- Programmer's Niche: nifty hints for programming in R (or S)
- Applications: Examples of analyzing data with R

The newsletter as a medium for communication intends to fill the gap between the R mailing lists and scientific journals: Compared with emails it is more persistent, one can cite articles in the newsletter and because the newsletter is edited it has better quality control. On the other hand, when compared to scientific journals, it is faster, less formal, and last but not least focused on R.

As all of R, R News is a volunteer project. The editorial board currently consists of the R core development team plus Bill Venables. We are very happy that Bill—one of *the* authorities on programming the S language—has offered himself as editor of "Programmer's Niche", a regular column on R/S programming.

This first volume already features a broad range of different articles, both from R core members and other developers in the R community (without whom R would never have grown to what it is now). The success of R News critically depends on the articles in it, hence we want to ask all of you to submit to R News. There is no formal reviewing process yet, however articles will be reviewed by the editorial board to ensure the quality of the newsletter. Submissions should simply be sent to the editors by email, see the article on page 30 for details on how to write articles.

We really hope you will enjoy reading R News!

Kurt Hornik
Technische Universität Wien, Austria
Kurt.Hornik@ci.tuwien.ac.at

Friedrich Leisch
Technische Universität Wien, Austria
Friedrich.Leisch@ci.tuwien.ac.at

Contents of this issue:

What is R?

A combined version of the files README and THANKS from the R source tree.

by the R Core Team

Introduction

R is "GNU S" — A language and environment for statistical computing and graphics. R is similar to the award-winning S system, which was developed at Bell Laboratories by John Chambers et al. It provides a wide variety of statistical and graphical techniques. R is free software distributed under a GNU-style copyleft.

The core of R is an interpreted computer language with a syntax superficially similar to C, but which is actually a "functional programming language" with capabilities similar to Scheme. The language allows branching and looping as well as modular programming using functions. Most of the user-visible functions in R are written in R, calling upon a smaller set of internal primitives. It is possible for the user to interface to procedures written in C, C++ or FORTRAN languages for efficiency, and also to write additional primitives.

The R distribution contains functionality for a large number of statistical procedures. Among these are: linear and generalized linear models, nonlinear regression models, time series analysis, classical parametric and nonparametric tests, clustering and smoothing. There is also a large set of functions which provide a flexible graphical environment for creating various kinds of data presentations.

A package specification allows the production of loadable modules for specific purposes, and several contributed packages are made available through the CRAN sites (see the article on *"R Resources"* in this issue).

History

R was initially written by Robert Gentleman and Ross Ihaka of the Statistics Department of the University of Auckland. In addition, a large group of individuals has contributed to R by sending code and bug reports.

Since mid-1997 there has been a core group who can modify the R source code CVS archive. The group currently consists of

> Douglas Bates, John Chambers, Peter Dalgaard, Robert Gentleman, Kurt Hornik, Ross Ihaka, Friedrich Leisch, Thomas Lumley, Martin Maechler, Guido Masarotto, Paul Murrell, Brian Ripley, Duncan Temple Lang and Luke Tierney.

Present status

The present version implements most of the functionality in the 1988 book "*The New S Language*" (the "Blue Book") and many of the applications. In addition, we have implemented a large part of the functionality from the 1992 book "*Statistical Models in S*" (the "White Book").

All the R functions have been documented in the form of help pages in an "output independent" form which can be used to create versions for HTML, LaTeX, text etc. An 800+ page Reference Index (a collection of all the help pages) can be obtained in a variety of formats. The document "*An Introduction to R*" provides a more user-friendly starting point, and there is an "*R Language Definition*" manual and more specialized manuals on data import/export and extending R. See the file 'INSTALL' in the R sources for instructions on how to generate these documents.

Goals

Our aim at the start of this project 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. Somewhat later, we started to turn R into a "real" system, but unfortunately we lost a large part of the efficiency advantage in the process, so have recently revised the memory management mechanism and are looking for other candidates for optimization.

Longer-term goals include to explore new ideas: e.g., virtual objects and component-based programming, and expanding the scope of existing ones like formula-based interfaces. Further, we wish to get a handle on a general approach to graphical user interfaces (preferably with cross-platform portability), and to develop better 3-D and dynamic graphics.

Thanks

R would not be what it is today without the invaluable help of these people, who contributed by donating code, bug fixes and documentation:

> Valerio Aimale, Thomas Baier, Ben Bolker, Göran Broström, Saikat DebRoy, Lyndon Drake, Paul Gilbert, Robert King, Kjetil Kjernsmo, Philippe Lambert, Jim Lindsey, Patrick Lindsey, Catherine Loader, Gordon Maclean, John Maindonald, Duncan Murdoch, Jens Oehlschlägel-Akiyoshi, Steve Oncley, Richard O'Keefe,

Hubert Palme, Jose C. Pinheiro, Martyn Plummer, Jonathan Rougier, Heiner Schwarte, Bill Simpson, Adrian Trapletti, Terry Therneau, Bill Venables, Gregory R. Warnes and Andreas Weingessel.

We have probably omitted some important names here because of incomplete record keeping. If we have overlooked you, please let us know and we'll update the list. Many more, too numerous to mention here, have contributed by sending bug reports and suggesting various improvements.

A special debt is owed to John Chambers who has graciously contributed advice and encouragement in the early days of R and later became a member of the core team.

R Resources

by the R Core Team

Frequently Asked Questions

A collection of Frequently Asked Questions (FAQs) and their answers is maintained by Kurt Hornik and can be found at the URL http://www.ci.tuwien.ac.at/~hornik/R/R-FAQ.html.

A text version is in file 'FAQ' in the top directory of the R sources, and an HTML version is available via the on-line help (on the index page given by `help.start()`).

Archives

The Comprehensive R Archive Network (CRAN) is a collection of sites which carry identical material, consisting of the R distribution(s), the contributed extensions, documentation for R, and binaries.

The CRAN master site (in Vienna, Austria) can be found at the URLs

http://cran.r-project.org/
ftp://cran.r-project.org/pub/R/

and is mirrored daily at many sites: see the list at http://cran.r-project.org/mirrors.html.

Mailing Lists

Thanks to Martin Maechler there are number of mailing lists which are used by R users and developers. They are

r-announce@lists.r-project.org:
: announcements of new R releases or applications.

r-help@lists.r-project.org:
: general inquiries and discussion about R.

r-devel@lists.r-project.org:
: discussions about the future of R and pre-testing of new versions.

To subscribe (or unsubscribe) to these mailing list send '`subscribe`' (or '`unsubscribe`') in the *body* of the message (not in the subject!) to

r-announce-request@lists.r-project.org
r-help-request@lists.r-project.org
r-devel-request@lists.r-project.org

Browsable archives of the mailing lists can be found at http://www.r-project.org/. Text files (in Unix mail folder format) of the archives are made available monthly; see the 'doc/mail/mail.html' file on any CRAN node.

Bug-tracking System

R has a bug-tracking system (or perhaps a bug-filing system is a more precise description) available on the net at

http://bugs.r-project.org/

and via e-mail to r-bugs@r-project.org. The R function `bug.report()` can be used to invoke an editor from within an R session and send the report to the right address. It also fills in some basic information, such as your R version and operating system, which has proved helpful in the debugging process.

The source distribution has a file 'BUGS' at the top level giving a summary of the entries at the time the R distribution was prepared.

Changes in R

by the R Core Team

New features in version 1.2.1

- Package **mva** has new functions `factanal()`, `varimax()`, `promax()`, and examples.
- New functions `readBin()` and `writeBin()` to transfer binary data to and from connections.
- `merge()` is partially moved to C to reduce its memory usage.
- `library(help = PKG)` now displays the contents of the package's 'DESCRIPTION' file in addition to its 'INDEX'.
- `Sd2Rd` can handle S4-style documentation too: see *"Writing R Extensions"*.
- `prompt()` now also works with a character argument (useful for producing many '*.Rd' files in a loop).
- The Unix front-end shell script now ignores a value for `R_HOME` found in the environment.
- Connections functions such as `file()` now accept a description of length > 1, with a warning.
- All text-mode connections now accept input with LF, CR or CRLF line endings. This means that `readLines()` can be used on DOS files and `source()` on Mac files, for example.

  Also, CRLF-terminated files can be used as stdin on Unix, and files with last lines without an EOL mark can be used as stdin and `source()`-ed on Unix and Windows.
- 'DESCRIPTION' file has a new recommended '`Maintainer:`' field.
- `stars()` now uses a larger 'cex' for the labels, and 'cex' and 'lwd' are now arguments. Further, the argument names (xlim, ylim, axes) are now consistent with other plot functions. The key symbol is not clipped anymore into the plot region by default.
- Date-time quantities are now printed with the timezone, if known.
- `R CMD build` now ignores all files specified (via Perl regexps) in file '.Rbuildignore' in the top-level source directory of a package.
- Horizontal boxplots are possible with 'horizontal = TRUE'.
- `all.equal()` on lists now compares them as generic vectors, that is they are equal if have identical names attributes and all components are equal.
- Invalid lines in '.Renviron' now give warnings when R is started.
- Argument 'na.last' implemented for `rank()`.

New features in version 1.2.0

- There is a new memory management system using a generational garbage collector. This improves performance, sometimes marginally but sometimes by double or more. The workspace is no longer statically sized and both the vector heap and the number of nodes can grow as needed. (They can shrink again, but never below the initially allocated sizes.) See `?Memory` for a longer description, including the new command-line options to manage the settings.
- values of '`--min-nsize`' up to 50M (2Gb on 64-bit Solaris) are allowed.
- A (preliminary) version of S4-like connections has been added, and most functions which take a 'file' argument can now work with a connection as well as a file name. For more details, see the chapter on Connections in the *"R Data Import/Export"* manual.
- New command-line option '`--no-restore-history`' implied by '`--vanilla`'.
- Command-line option '`--no-restore`' is now '`--no-restore-data`' and '`--no-restore`' implies '`--no-restore-*`' (currently '`data`' and '`history`').
- The more recent GNU regex from grep-2.4.2 is used. This uses locale-based ordering for ranges on platforms with `strcoll`.
- The print routines now escape '`"`' (as '`\"`') in a character string only when it is printed as a quoted string. This makes `print(, quote=FALSE)` and `cat()` consistent.
- The standard methods for `add1()` and `drop1()` now attempt to cope with missing values by using a subset of the data that is "cleaned" by `na.action` for the maximal model under consideration.

- `anova()` for 3 or more lm objects now behaves compatibly with S and `anova.glmlist()`. The old behaviour is still available by calling `anovalist.lm()` directly.

- `anova()` for multiple lm and glm objects no longer truncates the formula printed. There is much more extensive documentation for `anova()` methods.

- New method `as.data.frame.table()` for converting the array-based representation of a contingency table to a data frame containing the classifying factors and the corresponding counts.

- New function `assocplot()` for producing Cohen-Friendly association plots.

- `autoload()` accepts 'lib.loc' and other arguments to `library()`.

- `bxp()` has new argument 'frame.plot', as `plot.default()`.

- `contour()` now has 'axes' and 'frame.plot' args.

- `contrasts(, FALSE)` now always returns an identity matrix, to make `model.matrix` compatible with S. This affects models such as `lm(y ~ o - 1)` where `o` is an ordered factor.

- 'where' command added to `debug()`.

- `demo(dynload)` (which used the superseded `call_R` interface) has been removed.

- Class `"dendrogram"` in package **mva** providing general support for tree-like structures (plotting, cutting, ...).

- `dev.copy2eps()` and `dev2bitmap()` preserve the aspect ratio of the copied device if just one of 'width' and 'height' is specified.

- `dump()` has new argument 'append', argument 'fileout' has been renamed to 'file' (for consistency with all other functions).

- `edit.default()` now checks for an unset 'editor' argument, and terminates with an error if the editor cannot be run.

- The 'mode' argument of `exists()` and `get()` is interpreted as `mode(x)` rather than `typeof(x)`, following S.

- New functions `file.access()` and `file.info()` for information on files on the user's file systems.

- New convenience function `file.copy()`.

- `file.show()` allows 'pager' argument to be an R function, and consequently, the 'pager' option can be an R function.

- Formatting (and printing) of data.frames with complex objects is improved. `toString()` was added as a new function.

- `format()` has a new argument 'justify' controlling the justification of character strings (and factors).

- Formula objects now have an environment and code manipulating them needs to take care to preserve it or set an appropriate environment.

- New function `fourfoldplot()` for producing fourfold displays of 2 by 2 by k contingency tables.

- `gc()` now reports the space allocated, not the space free, since the total space is now variable.

- New primitive `gc.time()` to report on time spent in garbage collection.

- `hclust()` takes new argument 'members' allowing dissimilarity matrices both for singletons (as until now) and clusters.

- `help()` has an additional 'pager' argument which may be passed to `file.show()` (useful for ESS fans).

- There is now an R 'Hershey' list object for Hershey vector font computations and documentation.

- `hist()` now returns an object of class `"histogram"` and calls the new function `plot.histogram()` for plotting. It now also allows character labels.

- `if(*)` now gives a more intelligible error message when '*' cannot be coerced to logical.

- `inherits()` is now an internal function and compatible with S.

- New function `lag.plot()` in package **ts**.

- `legend()` has a new argument 'pt.bg'.

- The commands history can be loaded with `loadhistory()`, saved with `savehistory()` and displayed with `history()`, under Windows and under Unix using the readline or GNOME interfaces.

- `mad()` has new (logical) arguments 'low' and 'high' (the first giving S compatibility).

- New function `manova()` and summary method.

- Function `mantelhaen.test()` in package **ctest** now can deal with general $I \times J \times K$ tables. In addition, in the $2 \times 2 \times K$ case, it can also perform an exact conditional test of independence, and gives confidence intervals for the common odds ratio.

- `model.frame()` now uses the environment of its formula argument, rather than the parent environment, to evaluate variables not found in the data argument. See `help(formula)`.

- `mosaicplot()` can now also create extended mosaic plots, which visualize the residuals from a log-linear model using color and outline.

- New utility function `n2mfrow()`.

- `nlm(check.analyticals = TRUE)` now warns if the supplied gradient and/or hessian are of the wrong length.

- New function `object.size()` to give approximate memory allocation.

- `optim()` now checks the length of an analytical gradient at each evaluation.

- The L-BFGS-B method of `optim()` now support tracing, at several levels of detail.

- `options(check.bounds = TRUE)` makes each vector extension by sub-assignment produce a warning.

- `options(width)` now admits to a limit (previously 200, now 10000) and gives a more informative message if out of range (as it does now for digits and expressions).

- Function `path.expand()` to do tilde-expansion on file paths. This provides an interface to `R_ExpandFileName`, which is now a documented entry point.

- `.Platform` has new component `endian`, useful for binary file manipulations.

- `plot.function()` and `curve()` now take xlim as default for (from,to) if the former is specified.

- `plot.hclust()` allows arguments 'main', 'sub', etc., and has non-empty defaults for these.

- `plot.ts(x,y)` now allows to suppress labels and lines; it is better documented.

- The `postscript()` driver now allows a user-specified family so, for example, one can use the same fonts in diagrams as in running text.

- The `postscript()` driver allows its prolog to be changed (by an expert) via object `.ps.prolog`.

- `prop.table()` and `margin.table()` now work with an empty 'margin'.

- Formerly deprecated function `provide()` is now defunct.

- New functions `read.delim()`/`read.delim2()` to make it easier to read delimited files as Windows programs tend to create (usually TAB separated).

- New `readLines()` function to read a file line-by-line.

- New functions `reshapeLong()` and `reshapeWide()` emulating Stata's reshape command. These are still labeled experimental and might be improved (or removed) in later versions.

- `row.names()` and `row.names<-()` are now generic functions which call `rownames()` as their default method and have methods for class `"data.frame"`.

- New function `Rprof()` for profiling R expressions under Unix. Configure with '`--enable-R-profiling`' (on by default) to make this operational.

- `save(, oldstyle=TRUE)` has been withdrawn.

- `scan()` and `read.table()` have a new argument 'fill' which can be set `TRUE` to allow reading files with unequal number of fields per line. (Programs like Excel have a habit of creating such files when exporting.)

- `scan()` and `read.table()` have a new argument 'blank.lines.skip' to allow blank lines to be read.

- `scan()` now reads empty character fields as `""` not `"NA"` unless `""` is included in na.strings.

- `smooth()` in package **eda** has a better default (3RS3R instead of 3RSR) and more arguments, e.g., 'twiceit' for some S compatibility and 'kind = "3R"' for running medians of 3.

- `strsplit()` has a new argument 'extended' controlling whether to use extended (the default) or basic regular expressions for splitting.

- `Sys.getenv()` becomes the preferred name for `getenv()`, which is now deprecated.

- New functions `Sys.getlocale()` and `Sys.setlocale()` to query and set aspects of the locale of the R process, and `Sys.localeconv()` to find the default decimal point, etc.

- New function `Sys.info()` for platform, host and user information.
- New function `Sys.putenv()` to set environment variables.
- New function `Sys.sleep()` to suspend execution for a while.
- Date-time support functions with classes `"POSIXct"` and `"POSIXlt"` to represent dates and times (resolution 1 second) in the POSIX formats. Functions include `Sys.time()`, `as.POSIXct()`, `strftime()`, `strptime()`, and methods for `format`, `plot`, `c`, ... There are conversion functions for objects from packages **date** and **chron**; unlike those packages these support functions know about time zones (if the OS does).
- **tcltk** package now has `tkpager()` which is designed to be used by `file.show()` and shows help pages etc. in separate text widgets.
- **tcltk** is now more careful about removing the objects representing widgets in the R workspace when the windows are destroyed (e.g., using window manager controls)
- **tcltk** package has had several canvas functions implemented.
- **tcltk** now wraps callbacks to R in a `try()` construct—the nonlocal return from R's error handling could bring the Tk system into a strange state.
- New demos for **tcltk**: tkfaq, tkfilefind, tkcanvas.
- `termplot()` now has an 'ask' argument.
- `terms()` creates objects which now inherit from class `"formula"`, so for example `as.formula(terms.object)` needs to be replaced by `formula(terms.object)`.
- `traceback()` is now printed un-quoted and labelled by the frame number.
- New argument 'recursive' to `unlink()`. The default behaviour on Unix is now that of `rm -f`, not `rm -rf`. `unlink()` is now compatible across platforms.
- New functions `write.ftable()` and `read.ftable()` for writing out and reading in flat contingency tables.
- `write.table()` now quotes factor columns if 'quote=TRUE', and has a new argument 'qmethod' to control the escaping of embedded quotes in character or factor columns.
- New function `xtabs()` providing a formula interface to cross tabulation.
- The *"R Data Import/Export"* ('R-data.texi') manual has been added.
- The set of valid R names is now described (at last) in R-intro.
- The *"R Language Definition"* ('R-lang.texi') manual is now included and built in the same way as the other manuals.
- The R manuals (R-intro, R-exts, ...) are converted to HTML format (if the necessary Texinfo tools are available) and linked into the top HTML help page.
- The header file 'R.h' and those included from it are now usable with C++ code.
- New header file 'R_ext/Boolean.h': `Rboolean` type with `TRUE` and `FALSE` enum constants.
- New header file 'Rgraphics.h' to allow addons to use graphics structures.
- Recommended include file 'Rmath.h' replaces 'R_ext/Mathlib.h'.
- Bessel, beta and gamma functions are now documented as part of the API. Undocumented entry points are no longer in the header files, and some are no longer visible.
- `Calloc` and `Realloc` failures now give size information.
- 'DESCRIPTION' file in installed packages has a new '`Built:`' field giving build information (R version, platform, date).
- Much improved support for C++ code in add-on packages under Unix. New configure/build variables `SHLIB_CXXLD` and `SHLIB_CXXLDFLAGS` for specifying the command and flags needed for building shared libraries containing objects from a C++ compiler. Configure tries to get these right in typical cases (GNU tools and/or common platforms). C++ source suffixes '.cpp' and '.C' are now recognized in addition to '.cc'.
- Configure/build variables `MAINLD` and `MAINLDFLAGS` are renamed to `MAIN_LD` and `MAIN_LDFLAGS` for consistency with other `MAIN_*` variables, similarly for `SHLIBLD` and `SHLIBLDFLAGS`.
- Configure/build variable `FLIBS` now only contains the FORTRAN 77 intrinsic and run-time libraries needed for linking a FORTRAN 77 program or shared library (as determined by `configure`). BLAS library detection was extended, with results saved to the Make variable `BLAS_LIBS` which is also available to add-on packages.

- R CMD build and check have been completely re-written in Perl. In addition to running examples, check now also checks the directory structure and control files, makes a temporary installation and runs LaTeX on the help pages. build has been reduced to cleaning, rewriting indices and creating tar files.

  The same files of Perl code are now also used under Windows.

- Add-ons for utilities like Perl or LaTeX have now a central place in '$R_HOME/share'. Migration of existing files might take a while, though.

- Preliminary support for building R as a shared library ('libR') under Unix. Use configure with option '--enable-R-shlib' or do make libR in directory 'src/main' to create the shared library.

  There is also a linker front-end R CMD LINK which is useful for creating executable programs linked against the R shared library.

Changes on CRAN

by Kurt Hornik and Friedrich Leisch

Introduction

This column, named *Changes on CRAN*, will be one of the regular columns appearing in every volume of the newsletter. We will try to shortly summarize all changes on CRAN and the R web pages, list new or updated extension packages, new manuals, etc.

Split of CRAN and R homepage

During the second half of 2000 we have split up R's web pages into two seperate web sites:

http://www.r-project.org/
http://cran.r-project.org/

The first is meant as R's central homepage, giving information on the R project and everything related to it. The second—CRAN—acts as the download area, carrying the software itself, extension packages, PDF manuals; in short everything you may need to download for using R.

The main motivations for this "artificial" split were:

- CRAN is mirrored on a number of sites worldwide, hence we should try to keep it as small as possible to make life for the mirror sites easier. It should carry the material users of R need to download on a regular basis, such that having a mirror nearby pays off.

- We do not want to be (technically) limited in the possibilities how to present material on the homepage of R. However, a site like CRAN that is intended for other sites to mirror is very limited, because the administrators have no control of the mirror web servers. Hence, not even the simplest CGI scripts are possible, everything has to be hardcoded into physically existing HTML files.

Both sites are closely linked to each other, and we try to avoid duplicating information in as much as possible. In fact, http://www.r-project.org/ and http://cran.r-project.org/ are aliases for the same machine.

New CRAN mirrors

Laszlo Tornoci of the Semmelweis University Medical School in Budapest has set up a CRAN mirror for Hungary. Daniele Medri, with the Economics Faculty of University of Bologna, is working on setting up an Italian mirror.

Thus far, CRAN mirrors only exist in Europe and North America. In the interest of preserving bandwidth, we would like to add mirrors in other continents as well. Please contact wwwadmin@cran.r-project.org of you are interested in providing a new CRAN country mirror.

CRAN packages

CRAN contains R extension packages in four location:

'src/contrib': The main location containing packages that pass R CMD check at least on one platform[1].

'src/contrib/Devel': Packages that are under development, incomplete, do not pass R CMD check or where the authors think they are not ready for the main section.

[1]Debian GNU/Linux, the platform CRAN itself runs on and is used by the CRAN maintainers. The reason is a simple and practical one: before installing a package on CRAN we check it …

'src/contrib/Omegahat': Packages from the Omegahat project, see the article by John Chambers and Duncan Temple Lang on *"Omegahat Packages for R"*.

'contrib/extra': Packages that run only on a limited number of platform, e.g., because they are only available in binary format or depend on other software which is not free.

This section will list new or updated extension packages in 'src/contrib' since the last newsletter. Of course this is hard in the first volume, hence we decided to list all new packages since the release of R 1.1.0, which basically covers the second half of 2000:

Matrix A Matrix package for R, by Douglas Bates and Saikat DebRoy

PHYLOGR Functions for phylogenetically-based statistical analyses, by Ramón Díaz-Uriarte and Theodore Garland, Jr.

RODBC ODBC support and a back end database, by Michael Lapsley

RPgSQL PostgreSQL access, by Timothy H. Keitt

Rstreams Binary file stream support functions, by B. D. Ripley and Duncan Murdoch

XML XML parsing tools for R and S, by Duncan Temple Lang

conf.design Construction of factorial designs, by Bill Venables

dse Multivariate time series library, by Paul Gilbert

ellipse library ellipse for drawing ellipses and ellipse-like confidence regions, by Duncan Murdoch and E. D. Chow (porting to R by Jesus M. Frias Celayeta)

exactRankTests Exact distributions for rank tests, by Torsten Hothorn

foreign Read data stored by Minitab, SAS, SPSS, Stata, ..., by Thomas Lumley, Saikat DebRoy and Douglas M. Bates

gafit Genetic algorithm for curve fitting, by Telford Tendys

gld Basic functions for the generalised (Tukey) lambda distribution, by Robert King

maptree Mapping and graphing tree models, by Denis White

mgcv Multiple smoothing parameter estimation and GAMs by GCV, by Simon Wood

muhaz Hazard function estimation in survival analysis, S original by Kenneth Hess, R port by Robert Gentleman

mvnmle ML estimation for multivariate normal data with missing values, by Kevin Gross

mvtnorm Multivariate normal and t distributions, by Alan Genz and Frank Bretz, R port by Torsten Hothorn

scatterplot3d 3D scatter plot, by Uwe Ligges

sn The skew-normal distribution, by Adelchi Azzalini

splancs Spatial and space-time point pattern analysis, by Barry Rowlingson and Peter Diggle, adapted and packaged for R by Roger Bivand

tensor Tensor product of arrays, by Jonathan Rougier

wle Weighted likelihood estimation, by Claudio Agostinelli

xtable Export tables, by David Dahl

New Linux packages

The 'bin/linux' subdirectory of every CRAN site now also contains Mandrake 7.2 i386 packages by Michele Alzetta. In addition, RedHat 7.0 packages for the alpha and i386 platforms and SuSE 7.0 i386 packages were added. These packages are maintained by Naoki Takebayashi, Martyn Plummer, and Albrecht Gebhardt, respectively.

The Debian GNU/Linux packages can now be accessed through APT, the Debian package maintenance tool. After adding the line

```
deb \
  http://cran.r-project.org/bin/linux/debian \
  distribution main
```

(where *distribution* is either 'stable' or 'unstable'; feel free to use a CRAN mirror instead of the master) to the file '/etc/apt/sources.list', the programs `apt-get`, `apt-cache`, and `dselect` (using the apt access method) will automatically detect and install updates of the R packages.

Kurt Hornik
Technische Universität Wien, Austria
Kurt.Hornik@ci.tuwien.ac.at

Friedrich Leisch
Technische Universität Wien, Austria
Friedrich.Leisch@ci.tuwien.ac.at

Under New Memory Management

by Luke Tierney

R 1.2 contains a new memory management system based on a generational garbage collector. This improves performance, sometimes only marginally but sometimes by a factor of two or more. The workspace is no longer statically sized and both the vector heap and the number of nodes can grow as needed. They can shrink again, but never below the initially allocated sizes.

Generational Garbage Collection

Garbage collection is the colorful name usually used for the process of reclaiming unused memory in a dynamic memory management system. Many algorithms are available; a recent reference is Jones and Lins (1996). A simple garbage collection system works something like this: The system starts with a given amount of heap memory. As requests for chunks of memory come in, bits of the heap are allocated. This continues until the heap is used up. Now the system examines all its variables and data structures to determine which bits of allocated memory are still in use. Anything that is no longer in use is garbage (hence the name) and can be reclaimed for satisfying the next set of allocation requests.

The most costly step in this process is determining the memory still in use. Generational collection, also called ephemeral collection, is designed to speed up this process. It is based on the observation that there tend to be two kinds of allocated objects. Objects representing built-in functions or library functions and objects representing data sets being analyzed tend to be needed for long periods of time and are therefore present at many successive collections. Intermediate results of computations, on the other hand, tend to be very short lived and often are only needed for one or two collections.

A generational collector takes advantage of this observation by arranging to examine recently allocated objects more frequently than older objects. Objects are marked as belonging to one of three generations. When an object is initially allocated, it is placed in the youngest generation. When a collection is required, the youngest generation is examined first. Objects still in use are moved to the second generation, and ones no longer in use are recycled. If this produces enough recycled objects, as it usually will, then no further collection is needed. If not enough recycled objects are produced, the second generation is collected. Once again, surviving objects are moved to the next, the final, generation, and objects no longer in use are recycled. On very rare occasions even this will not be enough and a collection of the final generation is needed. However for most collections examining the youngest generation is sufficient, and collections of the youngest generation tend to be very fast. This is the source of the performance improvement brought about by the generational collector.

Limiting Heap Size

Since the size of the R workspace is adjusted at runtime, it is no longer necessary to specify a fixed workspace size at startup. However, it may be useful to specify a limit on the heap size as a precaution in settings where attempting to allocate very large amounts of memory could adversely affect other users or the performance of the operating system.

Many operating systems provide a mechanism for limiting the amount of memory a process can use. Unix and Linux systems have the `limit` and `ulimit` commands for `csh` and `sh` shells, respectively. The Windows version of R also allows the maximal total memory allocation of R to be set with the command line option '`--max-mem-size`'. This is analogous to using the `limit` command on a Unix system.

For somewhat finer control R also includes command line arguments '`--max-nsize`' and '`--max-vsize`' for specifying limits at startup and a function `mem.limits` for setting and finding the limits at runtime. These facilities are described in more detail in `?Memory`.

If R hits one of these limits it will raise an error and return to the R top level, thus aborting the calculation that hit the limit. One possible extension currently under consideration is a more sophisticated error handling mechanism that might allow the option of asking the user with a dialog whether the heap limits should be raised; if the user agrees, then the current calculation would be allowed to continue.

API Changes

Implementing the generational collector required some changes in the C level API for accessing and modifying internal R data structures.

For the generational collector to work it has to be possible to determine which objects in the youngest generations are still alive without examining the older generations. An assignment of a new object to an old environment creates a problem. To deal with this problem, we need to use a *write barrier*, a mechanism for examining all assignments and recording any references from old objects to new ones. To insure accurate recording of any such references, all assignments of pointers into R objects must go through special assignment functions. To allow the correct use of these assignment functions to be checked reliably by the C compiler, it was necessary to also require that all reading of pointer fields go through special accessor functions or macros. For example, to access element `i` of vector `x` you need to use

```
VECTOR_ELT(x, i)
```

and for assigning a new value `v` to this element you would use

```
SET_VECTOR_ELT(x, i, v)
```

These API changes are the main reason that packages need to be recompiled for 1.2. Further details on the current API are available in *"Writing R Extensions"*.

Future Developments

There are many heuristics used in the garbage collection system, both for determining when different generations are collected and when the size of the heap should be adjusted. The current heuristics seem to work quite well, but as we gain further experience with the collector we may be able to improve the heuristics further.

One area actively being pursued by the R core team is interfacing R to other systems. Many of these systems have their own memory management systems that need to cooperate with the R garbage collector. The basic tools for this cooperation are a finalization mechanism and weak references. A preliminary implementation of a finalization mechanism for use at the C level is already part of the collector in R 1.2. This will most likely be augmented with a weak reference mechanism along the lines described by Peyton Jones, Marlow and Elliott (1999).

References

Richard Jones and Rafael Lins (1996). *Garbage Collection*. Wiley. 10

Simon Peyton Jones, Simon Marlow, and Conal Elliott (1999). Stretching the storage manager: weak pointers and stable names in Haskell. http://www.research.microsoft.com/Users/simonpj/Papers/papers.html. 11

Luke Tierney
University of Minnesota, U.S.A.
luke@stat.umn.edu

On Exact Rank Tests in R

by Torsten Hothorn

Linear rank test are of special interest in many fields of statistics. Probably the most popular ones are the Wilcoxon test, the Ansari-Bradley test and the Median test, therefore all implemented in the standard package **ctest**. The distribution of their test statistics under the appropriate hypothesis is needed for the computation of P-values or critical regions. The algorithms currently implemented in R are able to deal with untied samples only. In the presence of ties an approximation is used. Especially in the situation of small and tied samples, where the exact P-value may differ seriously from the approximated one, a gap is to be filled.

The derivation of algorithms for the exact distribution of rank tests has been discussed by several authors in the past 30 years. A popular algorithm is the so called network algorithm, introduced by Mehta and Patel (1983). Another smart and powerful algorithm is the shift algorithm by Streitberg and Röhmel (1986). In this article, we will discuss the package **exactRankTests**, which implements the shift algorithm. The computation of exact P-values and quantiles for many rank statistics is now possible within R.

Using ExactRankTests

The algorithm implemented in package **exactRankTests** is able to deal with statistics of the form

$$T = \sum_{i=1}^{m} a_i$$

where $a = (a_1, \ldots, a_N)$ are positive, integer valued scores assigned to N observations. We are interested in the distribution of T under the hypothesis that all permutations of the scores a are equally likely. Many rank test can be regarded this way, e.g. the Wilcoxon test is a special case with $a_i = i$ and the Ansari-Bradley test has scores $a_i = \min(i, N - i + 1)$. For details about the algorithm we point to the original articles, for example Streitberg and Röhmel (1986) and Streitberg and Röhmel (1987).

Package **exactRankTests** implements the functions `dperm`, `pperm`, and `qperm`. As it is standard in R/S they give the density, the probability function and the quantile function. Additionally, the function `pperm2` computes two-sided P-values. Consider e.g. the situation of the Wilcoxon test. Let `x` and `y` denote two vectors of data, possibly tied. First, we compute the ranks over all observations and second we compute the Wilcoxon statistic, which is the sum over the ranks of the `x` sample.

```
R> ranks <- rank(c(x,y))
```

```
R> W <- sum(ranks[seq(along=x)])
R> pperm(W, ranks, length(x))
```

The one-sided P-value is computed in the last line of the example. In the absence of ties the results of [pq]perm and [pq]wilcox are equal. An exact version of wilcox.test is provided as wilcox.exact. The following example is taken from Mehta and Patel (1998). The diastolic blood pressure (mmHg) was measured on 11 subjects in a control group and 4 subjects in a treatment group. First, we perform the one-sided Wilcoxon rank sum test.

```
R> treat <- c(94, 108, 110, 90)
R> contr <- c(80, 94, 85, 90, 90, 90,
              108, 94, 78, 105, 88)
R> wilcox.exact(contr, treat,
                alternative = "less")

        Exact Wilcoxon rank sum test

data:  contr and treat
W = 9, point prob = 0.019, p-value = 0.05421
alternative hypothesis: true mu is less than 0
```

The one-sided P-value is 0.05421 which coincides with the P-value of 0.0542 given in Mehta and Patel (1998). Additionally, the probability of observing the test statistic itself is reported as point prob.

Usually, the distribution is not symmetric in the presence of ties. Therefore the two-sided P-values need additional effort. StatXact computes the two-sided P-value as 0.0989 and wilcox.exact returns:

```
R> wilcox.exact(contr, treat)

        Exact Wilcoxon rank sum test

data:  contr and treat
W = 9, point prob = 0.019, p-value = 0.0989
alternative hypothesis: true mu is not equal to 0
```

Real or rational scores

The original algorithm is defined for positive, integer valued scores only. Streitberg and Röhmel (1987) suggested to approximate the distribution of a statistic based on real or rational scores by taking the integer part of the appropriately multiplied scores. A bound for the maximal possible error on the quantile scale can be derived (see the documentation of [dpq]perm for more details). As an example, we want to calculate the critical value for a two-sided van der Waerden test for samples of 10 untied observations each and a significance level of $\alpha = 0.05$:

```
R> abs(qperm(0.025, qnorm(1:20/21), 10))
[1] 3.872778
```

By default, the tolerance limit is set to tol=0.01 which means that the computed quantiles does not differ more than 0.01 from the true ones. Due to memory limitations, it might not be possible to calculate a quantile in such a way.

Another approach is to use integer scores with the same shape as the original ones. This can be achieved by mapping the real or rational scores into $\{1, \dots, N\}$. The idea behind is that one is not interested in approximating the quantiles but to have a test with the same properties as the original one. Additionally, the computational effort is the same as for the Wilcoxon test. This procedure was suggested by my colleague Berthold Lausen during a discussion about the shift algorithm. The two-sided P-value for the van der Waerden test of two samples x and y is now computed as follows:

```
R> N <- length(c(x,y))
R> sc <- qnorm(rank(c(x,y))/(N+1))
R> sc <- sc - min(sc)
R> sc <- round(sc*N/max(sc))
R> X <- sum(sc[seq(along=x)])
R> p <- pperm2(X, sc, length(x))
```

Conclusion

Using the newly introduced package **exactRankTests**, R users are able to compute exact P-values or quantiles of linear rank tests based on the Streitberg-Röhmel shift algorithm. The use the the procedures [dpq]perm is illustrated by many examples in the help files. Additionally, a modified version of wilcox.test using the exact procedures is provided as wilcox.exact. The performance of [dpq]perm is not as good as that of [pq]wilcox but should be sufficient for most applications.

References

Cyrus R. Mehta and Nitin R. Patel. A network algorithm for performing fisher's exact test in $r \times c$ contingency tables. *Journal of the American Statistical Association*, 78(382):427–434, June 1983. 11

Cyrus R. Mehta and R. Patel, Nitin. *StatXact-4 for Windows*. Cytel Software Cooperation, Cambridge, USA, 1998. 12

Bernd Streitberg and Joachim Röhmel. Exact distributions for permutations and rank tests: An introduction to some recently published algorithms. *Statistical Software Newsletters*, 12(1):10–17, 1986. 11

Bernd Streitberg and Joachim Röhmel. Exakte Verteilungen für Rang- und Randomisierungstests im allgemeinen c-Stichprobenfall. *EDV in Medizin und Biologie*, 18(1):12–19, 1987. 11, 12

Torsten Hothorn
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
Torsten.Hothorn@rzmail.uni-erlangen.de

Porting R to the Macintosh

by Stefano M. Iacus

In late July of last year, Ross Ihaka kindly gave me the code he used to build version 0.64 of R on the Macintosh while the current R release passed version 1.0. So some of the newest code didn't fit these old sources and the Macintosh device needed also an update due to the migration of MacOS towards MacOS X. In any case, without that initial starting point, there probably wouldn't now be a beta release of "R for Mac" 1.2.1 that is quite stable and equivalent to different platform binaries.

What is the actual state of this port? Probably, starting from one of the 1.2.x releases the Macintosh code will be included in the standard distribution of the sources as well as the binaries. The currently available release of binaries and sources for MacOS is 1.2.1 and can be found starting from the R Developer Page (`http://developer.r-project.org/`), as to say, it is still a beta developer version.

What is implemented that was not in the past? Actually, one of the most important things is the support for dynamic libraries[2] on which most of the contributed packages are based. Also the XDR save/load format has been implemented so that Macintosh users can exchange '`.RData`' session files with other platforms having this ability (i.e., Windows systems). Also temporary files, the help (`man`-like) engine, and the file I/O routines needed for the Postscript, PicTex and XFig devices work correctly. Many minor bugs have been fixed and some further functionality has been added like the metric information technology to write mathematical expressions on plots.

Environment variables actually do no exist under MacOS systems as well as the shell. Much functionality based on these two has been partially suppressed, other such functions have been implemented using system-level interactions with the Finder (a sort of window manager under MacOS). Some of the environment variables are stored in the preference file of the R Application.

Luckily, the memory manager has changed in version 1.2.0 of R and due to this fact the current release of R for MacOS ignores the size settings. The R task can use as much memory as it likes till the memory space reserved to the application by the user is full. The user simply needs to change memory parameters using a Finder/Information procedure as she does for all other applications under MacOS.

What is it left to do? The Tcl support and interactions with other applications are in the works. These imply the interactions with Emacs-like editors and the simulation of the `system` command under other systems. One other important need for R for Macintosh is the migration toward MacOS X. One first step is to substitute completely all the former system's API with the new Carbon API provided by Apple. This will allow a Macintosh machine with System 8.6 or greater to run applications either under the standard MacOS and under the new MasOS X.

MacOS X will soon be released by Apple. This system can be viewed as a Unix-like environment with a window manager that looks like the former Macintosh Finder. Under MacOS X the environment variables, different shells, Tcl/Tk support and many other utilities are implemented.

One drawback of MacOS X is that it needs very powerful and new-generation machines to run. The temptation to simply recompile R sources for Unix under MacOS X and to write a specific MacOSX device is always there but this will imply to abandon most of the Macintosh users to their fate without R.

So, in my opinion, migration to MacOS X can be implemented in two steps: the current one that simply tries to make R a native application both under MacOS Systems $\leq$ 9.04 and MacOS X using the Carbon Support by Apple. At the same time taking care for specific MacOS X facilities when available and write a MacOSX device parallel to the current Macintosh one. At least for one year or so this should be a good strategy, then a complete abandon of pre-MacOS X systems will be acceptable for the Macintosh audience.

Currently Macintosh specific code is handled using `#ifdef macintosh` and the Macintosh driver is simply implemented by the `Macintosh()` function. A not so bad idea, could be to use `#ifdef macosx` and `MacOSX()` respectively to take advantage of the new MacOS X.

There is really a great need for R for the Macintosh. Since the first public pre-alpha release of R 1.0, I've received a lot of feedback from users. This feedback has been invaluable. I would like to thank particularly A. Antoniadis, R. Beer, G. Sawitzki and G. Janacek among the others, and of course Luke, Martin, Kurt, Brian and Guido from the R Team helping me with different hints, tips and tricks along with C-code.

Stefano M. Iacus
Università degli Studi di Milano, Italy
`stefano.iacus@unimi.it`

[2]for `dyn.load`ing compiled C or Fortran functionality

The density of the non-central chi-squared distribution for large values of the noncentrality parameter

by Peter Dalgaard

Introduction

On December 14th, 2000 Uffe Høgsbro Thygesen reported on the R-help mailing list that the `dchisq` function was acting up when being passed moderately large values of the `ncp=` argument.

The code he used to demonstrate the effect was essentially

```
testit <- function(mu) {
    x <- rnorm(100000, mean=mu)^2
    hist(x, breaks=100, freq=FALSE)
    curve(dchisq(x, 1, mu^2), add=TRUE)
}
par(mfrow=c(2,1), mex=0.7)
testit(10)
testit(15)
```

This led to the display in figure 1. Further experimentation showed that the density was losing mass visibly when `mu` was about 13 and deteriorating rapidly thereafter.

The definition of the non-central χ^2 used in R is

$$f(x) = e^{-\lambda/2} \sum_{i=0}^{\infty} \frac{(\lambda/2)^i}{i!} f_{n+2i}(x) \tag{1}$$

where f_n is the density of the central χ^2 on n degrees of freedom. The coefficients to $f_{n+2i}(x)$ are the point probabilities of the Poisson distribution with parameter $\lambda/2$.

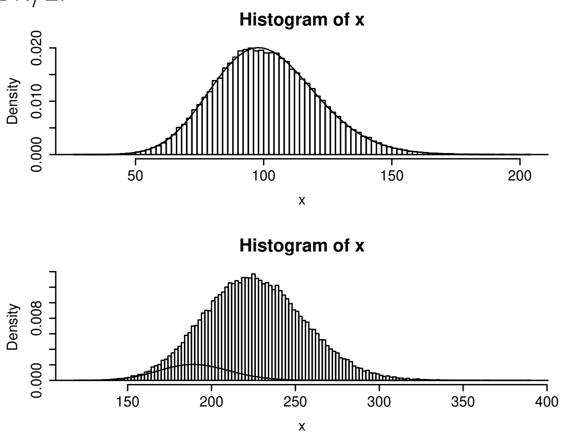

Figure 1: Demonstration of problem with old code for non-central χ^2. Top plot is for $\lambda = 10^2$, bottom one is for $\lambda = 15^2$.

A look at the source code in 'src/nmath/dnchisq.c' quickly revealed the source of the problem:

```
double
dnchisq(double x, double df, double lambda,
        int give_log)
{
    const static int maxiter = 100;
    ...
```

In the code, `maxiter` gives the truncation point of the infinite series in the definition. The author must have thought that "100 iterations should be enough for everyone", but at $\lambda = 225$ the Poisson weights will have their maximum for a value of i of approximately $225/2$ and the other part of the term is not small either: The mean of a non-central χ^2 distribution is $n + \lambda$, so $f_{n+2i}(x)$ with $i \approx \lambda/2$ is not small for relevant values of x. A quick display of the first 201 terms in the series can be obtained with the following code leading to Figure 2

```
i <- 0:200
plot(dpois(i, 225/2) * dchisq(225, 1+2*i),
     type='h')
```

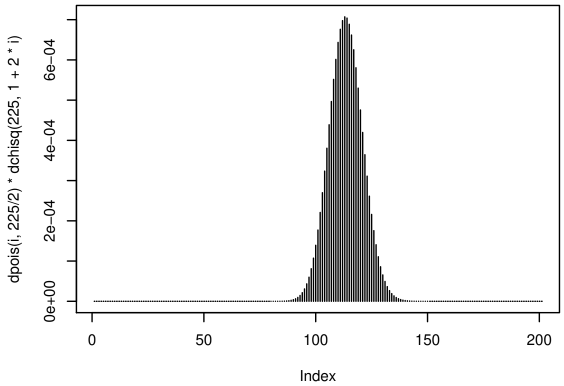

Figure 2: Terms of the series expansion for $\lambda = 225$ and $x = 225$

Obviously, truncating the series at $i = 100$ is not a good idea if one intends to be able to cover even moderately large values of the noncentrality parameter. However, although increasing `maxiter` to 10000 removed the problem for the arguments in the original report, the result was disappointing since it turned out that the modified routine would give a zero density already when `mu` was above 40. In what follows, I shall show what causes this effect and how to eliminate the problem.

Recurrence relations

Coding the series expansion (1) as written would be quite inefficient because of the large number of calculations of (central) χ^2 density values. Instead, one makes use of the recurrence relation

$$f_{n+2} = \frac{x}{n} f_n \tag{2}$$

which is easily derived from the definition of the χ^2 density

$$f_n(x) = \frac{1}{2\Gamma(n/2)} (x/2)^{n/2-1} e^{-x/2}$$

Similarly, the point probabilities for the Poisson distribution satisfy

$$p_{i+1} = \frac{\lambda}{i+1} p_i \tag{3}$$

Piecing (2) and (3) together one gets that if the terms of (1) are denoted a_i, then

$$a_{i+1} = \frac{\lambda x/2}{(i+1)(n+2i)} a_i \tag{4}$$

The code for the `dnchisq` C routine used this relation starting from

$$a_0 = e^{-\lambda/2} f_n(x)$$

However, when λ is large, this goes wrong because the interesting values of x are on the order of λ and $f_n(x)$ in the initial term will underflow the floating point representation:

```
> dchisq(40^2,1)
[1] 0
```

In those cases, the recurrence never gets started, $a_i = 0$ for all i.

Rearranging the recurrence

It is possible to get around the underflow problem by using the `give_log` argument to the C `dchisq` function, but the fact remains that most of the terms in the summation are effectively zero when λ is large.

It would be advantageous to calculate the summation "inside-out", i.e., start in the middle of the distribution of terms and proceed in both directions until the terms are too small to make any difference.

It is easy to find the value of i that gives the largest term by inspecting (4). The value of a_{i+1}/a_i will be less than 1 as soon as

$$\lambda x/2 < (i+1)(2i+n) \tag{5}$$

The right hand side is a second order expression in i and one easily finds that the roots of $(i+1)(2i+n) - \lambda x/2$ are

$$\frac{-(n+2) \pm \sqrt{(n-2)^2 + 4\lambda x}}{4} \tag{6}$$

and $i_{\max}$, the index of the maximum term is obtained by rounding the largest root upwards, unless that would result in a negative value in which case $i_{\max} = 0$ is the answer.

So we can start from $a_{i_{\max}}$ and use the recurrence relation (4) in both directions. Both when proceeding upwards and downwards, we can use the error bound obtained by dominating the series with a quotient series, since the terms are positive and the ratio between successive terms is decreasing in both directions. I.e. if the ratio of two successive terms is q (less than 1) then we know that the sum of the remainder of the series will be less than $\sum_1^\infty q^n = q/(1-q)$ times the current term and terminate the series if this is smaller than some preset value.

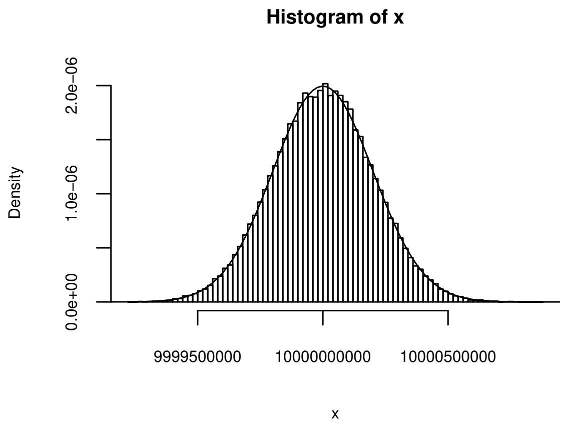

Figure 3: Result with new code for $\lambda = 100000^2$

The new code has been tested with values of λ as high as 100000^2 with good results (Figure 3), although it takes a while to complete for those values since there are on the order of a few times 100000 terms that must be included in the sum. At such high values of λ the noncentral χ^2 can of course be approximated extremely accurately by a Normal distribution.

Peter Dalgaard
University of Copenhagen, Denmark
`P.Dalgaard@biostat.ku.dk`

Connections

by Brian D. Ripley

Connections are a new concept in version 1.2.0 and still in an embryonic state, with some more facilities being made available in version 1.2.1. They are a far-reaching replacement for file-oriented input/output. They are modelled on the facilities introduced in S version 4 and described in Chambers (1998) but are likely to diverge away from that model as they become more central to R.

Three recent enquiries to `r-help` illustrate why one might want something more general than files. David Firth wanted to grab the output from an R function and send it via a socket to a machine in London (in connection with election forecasting as and when there comes a UK general election). He was happy to grab the output to a character vector, check it and then send it using `make.socket` and `write.socket`. Until version 1.2.0 the carefully formatted output of the standard statistical analysis functions could only be sent to the console or a file *via* `sink`. Using *output text connections*, `sink` can now 'trickle' output to a character vector.

Peter Kleiweg had a dataset which was comma-separated, but also had commas at the ends of lines. That might be good for human readers, but is not the format required by `scan`. My proffered solution was to use a *pipe connection* to pre-process the data file whilst it was being read into R, for example

```
zz <- pipe("sed -e s/,$// data")
res <- scan(zz, sep=",")
close(zz)
```

Erich Neuwirth wanted to remotely control R via a socket. That is, a read-mode socket could be created in an R session that would receive R expression as if typed at the command line and process each expression as soon as it was completed. We cannot do exactly that yet (although it was already planned), but one could set up a *pipe connection* to an external program (say `fromsocket`) to read from the socket and use a loop containing something like

```
zz <- pipe("fromsocket socket.id", "r")
repeat {
  ## read 1 expression
  eval(parse(file=zz, n=1))
  ## deal with other things here
  if(some condition) break;
}
close(zz)
```

although of course one needs to take care of error conditions, perhaps by using `try`.

The third example shows one of the main advantages of connections: they can be left open and read from (or written to) repeatedly. Previously the only way to repeatedly read from a file was to keep track of the number of rows read and use the `skip` argument to `scan` and allies. For writing, `cat` had an `append` argument to write to the end of a file. This was all tedious, and various functions (such as `sink` and `dump`) gained `append` arguments as users found a need for them. It was also expensive to keep on closing and re-opening files, especially if they were mounted from remote servers. There were other dangers, too. Suppose we were accessing a file and some process unlinked the file.[3] Then once the file is closed in R it will vanish. Unlinking the file could be accidental, but it is also a very sensible way to provide some protection, both against other processes interfering with it and against leaving it around after we have finished with it.

Types of connections

At present we have connections of types

file A text or binary file, for reading, writing or appending.

terminal There are three standard connections which are always available and always open, called `stdin`, `stdout` and `stderr`. These essentially refer to the corresponding C file names, so `stdin` is input from the console (unless input has been redirected when R was launched) whereas `stdout` and `stderr` are normally both output to the console.

pipe Used for either reading or writing. There is a special subclass for Windows GUI applications (not `rterm`) as the standard C pipes do not work there.

text An R character vector used as a text source.

output text A means to write text output to an R character vector, with each line forming a new element of the vector. Lines become visible as part of the vector in R immediately they are complete.

We envisage adding types. My scene-setting examples illustrated the potential usefulness of sockets as connections. There has been a proliferation of ways to access URIs,[4] such as `read.table.url`. Many (including this one) 'cheat' by downloading a file to a temporary location and then reading from that file. For `read.table`[5] that is not a bad solution,

[3] yes, that works under Unix at least.
[4] more commonly but less accurately known as URLs
[5] which currently needs to read the file twice

but it is desirable for `read.table` to be able to at least *appear* to read from connections that are URIs, and it looks desirable for R to have some basic abilities to read from `http` and `ftp` servers.

Good citizenship

A connection can exist and not be open, the life cycle of most types of connections being

$$\text{create} \longrightarrow \text{open} \longleftrightarrow \text{close} \longrightarrow \text{destroy}$$

A connection is created by its constructor, for example `file` or `pipe`, which can optionally open it. The 'good citizen' rule is that a function which finds a connection open should leave it open, but one that needs to open it should close it again. Thus the life cycle can include multiple passes across the double-sided arrow.

One has to be careful with the terminology here. With one exception, a function internally never destroys a connection, but the function `close` both closes (if necessary) and destroys one. The exception is `sink`, which does close and destroy the current sink connection, unless it is a terminal connection. There is no way for a user to explicitly close a connection without destroying it.

Text connections are special: they are open from creation until destruction. 'Closing' an output text connection flushes out any final partial line of output. (You can call `isIncomplete` to see if there is one.)

These semantics may change. R version 1.2.0 also introduced *references*, which allow finalization actions for referents once there is no R object referring to them, and this may be used to avoid the need for the explicit destroying of connections.

Leaving a connection closed but not destroyed is a minor waste of resources. (There is a finite set of connections and each takes a little memory.) One can find out about all the connections by `showConnections(all = TRUE)`, and there is a function `closeAllConnections` to close all close-able connections (that is, not the terminal connections).

What can I do with connections?

In short, almost everything you used to do with files. The exceptions are the graphics devices which write to files, and it is not yet clear if it is beneficial to be able to send graphics output to connections. The most obvious uses, to pipe the file to a printer, are already possible via other mechanisms.

There are new things made possible by the notion of leaving a connection open. For text-oriented applications, `readLines` and `writeLines` can read and write limited or unlimited amounts of lines from and to a character vector. Many connections are seekable (check this by `isSeekable`) and so can be restored to an arbitrary position.[6]

One can also for the first time work with binary files in base R.[7] Functions `readBin` and `writeBin` can read and write R vector objects (excluding lists) to a connection in binary mode. There are options to assist files to be transferred from or to other programs and platforms, for example to select the endian-ness and the size of the storage type.

Text *vs* binary

There is a distinction between text mode and binary mode connections. The intention is that text-based functions like `scan` and `cat` should use text mode connections, and binary mode is used with `readBin` and `writeBin`.

This distinction is not yet consistently enforced, and the main underlying difference is whether files are opened in text or binary mode (where that matters). It now looks as if opening all files in binary mode and managing the translation of line endings internally in R will lead to fewer surprises. Already reading from a connection in text mode translates lines endings from Unix (LF), DOS/Windows (CRLF) and Macintosh (CR) formats (and from all connections, not just files).

Looking forwards

I see connections as playing a more central rôle in future releases of R. They are one way to promote distributed computing, especially where a stream-oriented view rather than an object-based view is most natural.

The details are likely to change as we gain experience with the ways to use connections. If you rely on side-effects, it is well to be aware that they may change, and the best advice is to manage the connections yourself, explicitly opening and destroying them.

Reference

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

Brian D. Ripley
University of Oxford, UK
ripley@stats.ox.ac.uk

[6]at least if the underlying OS facilities work correctly, which they appear not to for Windows text files.
[7]Package **Rstreams** has provided a way to do so, and is still slightly more flexible.

Using Databases with R

by Brian D. Ripley

There has been a lot of interest in the inter-working of R and database management systems (DBMSs) recently, and it is interesting to speculate why now. Corporate information has been stored in mainframe DBMSs for a few decades, and I remember colleagues in the 1970s using SAS (via punched cards) to extract and process information from tapes written from such databases. I guess what has changed is accessibility: personal DBMSs are widely available and no longer need teams of experts to manage them (although they still help). The *R Data Import/Export* manual introduced in version 1.2.0 provides a formal introduction to most of the facilities available, which this article supplements by some further motivation, as well as allowing for personal views.

We can look at the interfaces from two viewpoints:

1. You have data you want to organize and extract parts of for data analysis. You or your organization may find it convenient to organize the data as part of a database system, for example to help ensure data integrity, backup and audit, or to allow several people simultaneously to access the data, perhaps some adding to it and others extracting from it.

   The need can be as simple as to have a data entry and verification system.

2. You want to do some simple data manipulations for which R is not particularly suitable. Someone on `r-help` wanted to do a merge on 30,000 rows. DBMSs are (usually) very good at merges, but R's `merge` is only designed for small-scale problems. Rather than spend time re-writing `merge`, why not use an already-optimized tool?

Choosing a DBMS

A wide range of DBMSs are available, and you may already have one in use. If not, the most popular contenders on Unix/Linux appear to be PostgreSQL (http://www.postgresql.org/) and MySQL (http://www.mysql.com/). PostgreSQL is Open Source and competitive on features and standards-conformance with the commercial big names (many of which have 'free' lightweight Linux versions). MySQL is 'lean and mean' but with limited security features, little transaction support, ...

On Windows, the most popular database of any sophistication is undoubtedly Access, which has a big brother SQL Server. Pre-compiled development versions of MySQL can be downloaded and were used for most of our testing under Windows. It is possible[8] to build and use PostgreSQL under the Cygwin environment.

I will assume that we are working with a *relational* database. These store the data in a collection of *tables* (also known as *relations*) which are closely analogous to R's data frames: they are conceptually rectangular made up of columns (or 'fields') of a single[9] type, for example character, monetary, datetime, real, ..., and rows ('records') for each case.

The common DBMSs are client-server systems, with a 'backend' managing the database and talking to one or more clients. The clients can be simple command-line interfaces[10] such as provided by `mysql` and `psql`, general-purpose or application-specific GUI clients, or R interfaces as discussed here. The clients communicate with the backend by sending requests (usually) in a dialect of a language called SQL, and receive back status information or a representation of a table.

Almost all of these systems can operate across networks and indeed the Internet, although this is often not enabled by default.

Choosing an interface

There are several ways to interface with databases. The simplest is the analogue of 'sneaker LAN', to transfer *files* in a suitable format, although finding the right format may well not be simple.

Four contributed R packages providing interfaces to databases are described in *R Data Import/Export*, but two are currently rather basic. By far the most portable option is **RODBC**, and unless you can choose your DBMS you probably have no other option. Open Database Connectivity (ODBC) is a standard originally from the Windows world but also widely available on Linux. It provides a common client interface to almost all popular DBMSs as well as other database-like systems, for example Excel spreadsheets. To use ODBC you need a driver manager for your OS and a driver for that and your DBMS. Fortunately drivers are widely available, but not always conforming to recent versions[11] of ODBC.

The basic tools of **RODBC** are to transfer a data frame to and from a DBMS. This is simple: use commands like

```
sqlSave(channel, USArrests, rownames = "state")
```

[8] but not easy, and I know of no pre-compiled distribution.
[9] with an important exception: see 'tips' below
[10] sometimes called 'monitors'
[11] that for MySQL is rather out of date.

```
sqlFetch(channel, "USArrests", rownames = TRUE)
```

to copy the R data frame to a table[12] in the database, or to copy the table to an R data frame. However, if we want to do more than use the database as a secure repository, we need to know more. One thing we can do is ask the DBMS to compute a subset of a table (possibly depending on values in other tables), and then retrieve the result (often known as a *result set*). This is done by `sqlQuery`, for example (all on one line)

```
sqlQuery(channel,
  "select state, murder from USArrests
   where rape > 30 order by murder")
```

which is the SQL equivalent of the R selection

```
z <- USArrests[USArrests$Rape > 30,
               "Murder", drop = FALSE]
z[order(z[,1]), drop = FALSE]
```

Indeed, almost anything we want to do can be done in this way, but there are shortcuts to a lot of common operations, including `sqlFetch` and `sqlSave`. As another example, let us perform the (tiny) `merge` example in a database.

```
sqlSave(channel, authors)
sqlSave(channel, books)
sqlQuery(channel,
  "SELECT a.*, b.title, b.otherauthor
   FROM authors as a, books as b
   WHERE a.surname = b.name")
```

This is of course pointless, but the technique it illustrates is very powerful.

There is a package[13] **RPgSQL** that provides a sophisticated interface to PostgreSQL. This provides analogues of the facilities described for **RODBC**. In addition it has the powerful notion of a *proxy data frame*. This is an object of an R class which inherits from data frames, but which takes little space as it is a *reference* to a table in PostgreSQL. There will be an advantage when we are accessing smaller parts of the data frame at any one time, when the R indexing operations are translated to SQL queries so that subsetting is done in PostgreSQL rather than in R.

Traps and tips

Things are not quite as simple as the last section might suggest. One problem is case, and one solution is only ever to use lowercase table and column names. As the mixtures of cases in the examples thus far suggest, many DBMSs have problems with case. PostgreSQL maps all names to lowercase, as does MySQL on Windows but not on Linux, Oracle maps all to uppercase and Access leaves them unchanged!

In a similar way the R column `other.author` was changed to `otherauthor` in the system used for the `merge` example.

Another problem is row-ordering. It is safest to regard the rows in a table as unordered, in contrast to a data frame, and indeed the optimization process used in executing queries is free to re-order the results. This means that we do need to carry along row names, and this is done by mapping them (more or less transparently) to a column in the table. It also explains why we sorted the results in the `sqlQuery` example.

Care is needed over missing values. The one exception to the rule that all entries in a column of a table must be of a single type is that an entry is allowed to be `NULL`. This value is often used to represent missing values, but care is needed especially as the monitors do not visually distinguish an empty character field from a `NULL`, but clients should. Michael Lapsley and I re-wrote **RODBC** to handle all the possibilities we envisaged.

We have so far given no thought to efficiency, and that is how it should be until it matters. Tuning databases is an art: this involves judiciously creating indices, for example. Hopefully research advances in database systems will percolate to more intelligent internal tuning by widely available DBMSs.

Another issue is to take care over is the size of the result set. It is all too easy to write a query that will create a result set that far exceeds not just the available RAM but also the available disc space. Even sensible queries can produce result sets too large to transfer to R in one go, and there are facilities to limit queries and/or transfer result sets in groups.

The future

Hopefully one day in the not too far distant future the R interfaces will be much more similar than at present, but that does depend on contributors looking at each other's designs.

It will be good to see more use of R's various types and classes, for example for times and monetary amounts.

There is a further fascinating possibility, to embed R inside a DBMS. Duncan Temple Lang mentions embedding R in PostgreSQL in a document[14] on the R developer's Web site. I don't have access to such a system, but his description is

> The initial example [...] was embedding R within PostgreSQL for use as a procedural language. This allows (privileged) users to define SQL functions as R functions, expressions, etc.

[12] probably named in lower case as here.
[13] which as far as I know it has only been used under Linux/Unix.
[14] http://developer.r-project.org/embedded.html

from which I gather that the (already rather extended) dialect of SQL can be extended by user-defined functions that call R to compute new columns from old ones.

We are exploring R–DBMS interfaces in data mining applications. There databases can be large but are perhaps growing slower than computing power. For example, insurance databases already cover around 30 million drivers. The idea is to use the DBMS not just to extract subsets for analysis but also perform cross-tabulations and search for exceptions.

Brian D. Ripley
University of Oxford, UK
ripley@stats.ox.ac.uk

Rcgi 4: Making Web Statistics Even Easier

by M.J. Ray

Webservers allow browsers to run a selection of programs chosen by the webmaster, via the Common Gateway Interface which defines how inputs are passed from browser to program and output passed back again. Naturally, it is possible to make R one of the available programs by using a "wrapper script" to translate between standard R input and outputs and CGI. Rcgi is such a script and has just been revised for a new, more powerful and easier-to-install release.

Benefits

Rcgi has been developed in response to a perceived need at the University of East Anglia. Even though we can provide students with their own copies of the R software for free, there are two principal reasons why it is still useful to provide access to our installation over the internet.

The first is that not all students have their own workstations. While this may change eventually, many students use the campus's open access computing facilities. Our department can manage software on only a small proportion of these machines, so for the others a web interface is simpler to access and insulates us from configuration changes beyond our control. As long as the computer can still run a web browser, they can use Rcgi.

Feedback from students on our third-year course (who were the first to use the system) suggests that the web front-end is popular partly because of its "batch mode" of operation, running some commands and returning the output, together with the commands for editing and resubmission.

The second and increasingly important benefit of Rcgi is the ability for the lecturer to provide worked examples to the students with the option of leaving spaces for the students to contribute their own data to the examples. Rather than having to write their own CGI programs for each example, lecturers need only write the HTML for the page (which many know already) and the program in the R language.

Example

Take the following snippet of R code, which defines a vector of numbers and generates the basic summary statistics for them:

```
test <- c(1,45,2,26,37,35,32,7,
          4,8,42,23,32,27,29,20)
print(summary(test))
```

which has the output

```
Min. 1st Qu  Median  Mean 3rd Qu.  Max.
1.00   7.75   26.50 23.13   32.75 45.00
```

For an elementary statistics course, the lecturer may wish to provide this as a worked example, but allow the students to change the values in the "test" list. The HTML code to do this is:

```
<form  method='post'
       action='/cgi-bin/Rcgi'>
<input type='hidden' name='script'
       value='test <- c(' />
<input type='text' name='script'
value='1,45,2,26,37,35,32,7,
       4,8,42,23,32,27,29,20' />
<input type='hidden'
       name='script' value=')
       print(summary(test))
       ' />
<input type='submit' value='go!' />
</form>
```

and a view of the data entry page and the results returned from Rcgi are included in figure 4. Hopefully by the time that this article appears, this example will be back online at the Rcgi site (address below).

Note that the code executed is displayed directly below the output, offering the student the chance to examine and modify the R program. Hopefully, they can learn from this experimentation in a similar way to having the entire system on their own computer.

The most commonly suggested point for improvement was the installation procedure, which was previously completely manual. The usual Unix `make` program is used to install the new release, with a script that attempts to locate the various

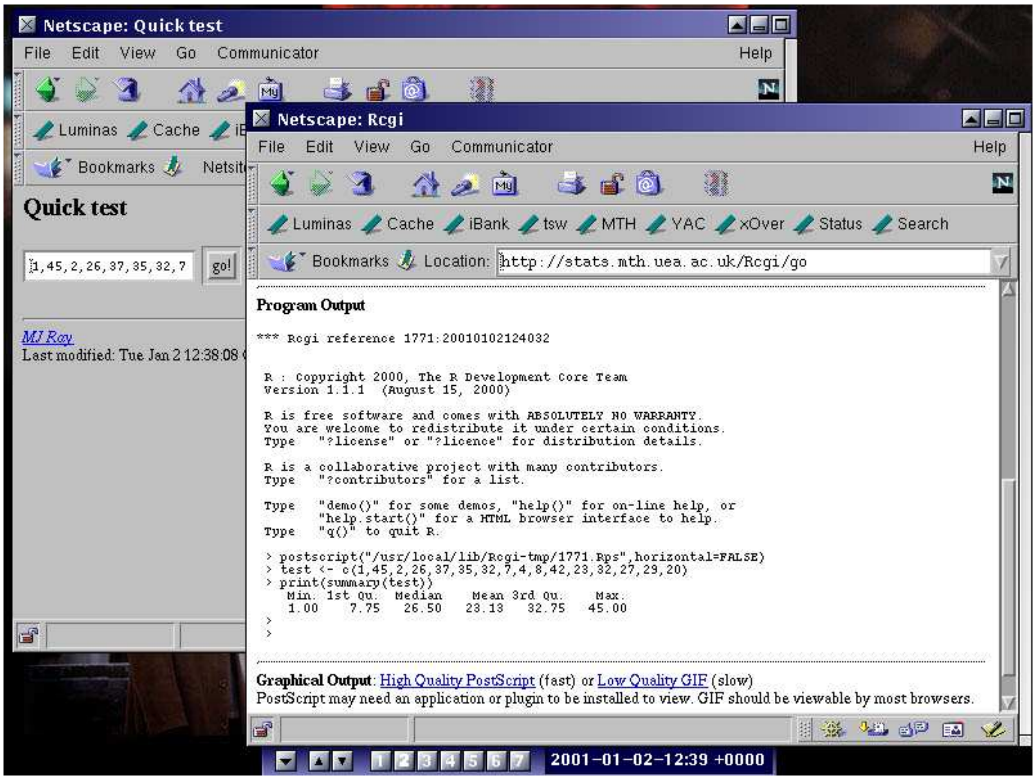

Figure 4: Screenshots of the results of the example HTML

programs it requires and build a module called 'Rcgi::SystemPrograms' detailing the location of some system programs required. The installer is still advised to review the scripts to make sure the details found are correct, though.

Other forthcoming improvements include better security checking (which is fairly basic at the moment) and better load balancing and session management. However, it is already a useful and popular system with a growing user community, with a mailing list available for help and development. The download site is http://stats.mth.uea.ac.uk/Rcgi/.

M.J. Ray
University of East Anglia, Norwich, UK
mjr@stats.mth.uea.ac.uk

Omegahat Packages for R

by John M. Chambers and Duncan Temple Lang

Overview

As part of the Omegahat project (http://www.omegahat.org/), we are developing a collection of packages to support new directions in programming in the S language, as implemented in R (http://www.r-project.org/) or in S-Plus (http://www.insightful.com/products/splus/). The packages we describe here illustrate ways to communicate between R and other languages and applications. You may find the packages useful if you want to access code from R that is written in another language, or call R functions from another programming language or application. This includes the notion of embedding R in such a package, or vice versa.

We also comment on future approaches to writing such software, with particular reference to making it available in both R and S-Plus.

We start with a very brief description of each of the available packages. More details can be found at the Omegahat web site (http://www.omegahat.org/). You can find a slightly lengthier description of how the packages relate to each other and the current tools in the sections below. We also plan to describe individual packages in future editions of the R newsletter. Please note that all of these packages are work-in-progress, at various different stages of evolution. We would be very appreciative of any comments on any of the topics.

The packages

There are currently five packages that provide different ways to communicate between R/S-Plus and other languages and applications:

XML facilities for reading and writing XML documents in S, allowing easier data exchange between S and other applications;

Java an interface for calling software written in Java from R and R functions from Java, and includes an R graphics device which uses the Java graphics facilities and can be customized using S functions;

RSPerl an interface for running Perl inside R, and R inside Perl so that one can create Perl objects, and invoke their methods and Perl subroutines from R, and vice-versa;

Python like the Perl and Java interfaces, this allows R functions to be called from Python scripts and Python classes, methods and functions to be called from R;

CORBA dynamic facilities for invoking methods in other applications, on other machines and written in a different language and also providing these types of server objects in S itself. In other words, this provides a high-level mechanism for distributed computing in S.

Inter-system interfaces

While S is a general programming language with which one *can* do almost anything, it is not always the most appropriate tool. Most of us will have encountered situations in which it is better to use other software, whether it be motivated by the need for efficiency or just accessing functionality already written in another language. The interfaces from S to subroutines in C or FORTRAN and to shell commands have been around a long time and provide the basic mechanisms to access other software. The new interfaces provide access to new languages. They also provide *better* interfaces, both in terms of how they communicate and also in the ability to embed the S language software in other systems. Embedding is important if we want to bring statistical software to the users of non-statistical systems.

The inter-system interfaces provide three basic ways to communicate between R or S-Plus and other applications. The first is to share data in a common format. We suggest using XML, the eXtensible Markup Language to do this. The next approach is to embed the application within R or vice-versa allowing direct communication within a single process housing the two systems. The **Java**, **RSPerl** and **Python** packages do this for each of these different languages. The last approach is to support inter-process, inter-machine communication and for this we use CORBA.

XML provides a way of specifying data so that it is self-describing and can be read by different applications from a single source. XML provides a way of saying what gets transferred between the applications, not how, be it files, URLs, sockets, databases, etc. The inter-language and CORBA interfaces provide the mechanism of how data is transferred, but focuses more on the exchange of functionality between the systems. We won't make further mention of the XML approach here as it is described in another article in this newsletter.

Perl, Python and Java

The Perl, Python and Java interfaces allow one to create objects, call functions and methods, and evaluate expressions in these languages as if they were local to R. A module or package written in any of these languages can be installed and loaded into the session without having to write any special code. Thus, you can use Perl's modules to access network services, Web facilities, operating system functions, etc. Similarly, one can use any of Python's rich set of modules and any package implemented in Java, including building graphical interfaces, communicating with databases, etc. Additionally, from R, we can even find out what modules, classes, routines and methods are available to us from these different systems and how to call them.

An immediate consequence of these interfaces is that one can replace calls to scripts in each of these languages with a direct call to evaluate an expression in Java, Python, or Perl. In other words, rather than calling a script written in any of these languages via the `system` function, we can call the language directly within the R session and have it execute that script. For example, we can use `.PerlExpr` to evaluate a Perl expression or `.PythonEvalFile` to execute a Python script.

Direct calls have some key advantages: we avoid starting a new process each time; results from earlier evaluations can be used in future evaluations; and objects are returned rather than lines of text containing these values. In fact, we can return *references* to objects, meaning that, say, the Java object can stay in Java rather than being converted and copied to R.

Let's consider a simple example. Using Java/Omegahat embedded inside R, we create a new window containing a button. We can then do some additional computations in R and return to Java to access the button and set the text it displays.

```
.OmegahatExpression("f = new
  GenericFrame(b = new JButton('No text'))")
  ...  # S code
.OmegahatExpression("b.setText('Get help')")
```

While using strings to send instructions to the other language may appear convenient at first, it is clumsy, error-prone and inefficient. Instead, it is better to invoke routines and methods directly from within the S language, passing the arguments as regular S objects, avoiding the awkward and limiting pasting of the arguments together to form the expression as a string. For example, we can create a network news reader, set the news group to read and get the first article with the S commands

```
news <- .PerlNew("News::NNTPClient")
msgNums <- news$group("comp.lang.python")[[1]]
news$article(as.integer(msgNums[[1]]))
```

The R-Perl interface handles invoking the corresponding methods in the NNTP object in Perl and converting both the arguments and the results to and from Perl.

This approach makes these foreign routines appear as if they are local S functions and leads to richer and more flexible results. The "magic" behind the interface is that we can refer to objects in these other languages directly as if they are local to S, and the interface handles the references appropriately. Clearly, in the Java example above, there is no sensible way to copy the button or the window/frame to an R object. In fact, we want to leave it in Java and manipulate it in subsequent R calls, such as

```
.Java(b, "setText", "Get Help")
```

Each of these interfaces also provides a mechanism for calling S code from that other language. This means that we can call code in that other language and have it callback to R during the execution of that code. Also, it means that users of those languages can access sophisticated statistical software in a way that is familiar to them without having to learn yet another language.

Inter-process communication: CORBA

The approach of embedding other applications and languages within the S session does not work easily for all systems. For example, sometimes we will want to run these other systems on a different machine. So, we need a way to access functionality in other processes to things such as retrieving or updating the data in a spreadsheet; allowing computations to be displayed in a GUI, etc.

The **CORBA** package (http://www.omegahat.org/RSCORBA/) provides this functionality, allowing users to call methods in remote objects as if they were local S functions. The remote objects can potentially be in a different application, running on a different machine and developed in a different programming language. Also, S users can offer facilities to other applications by creating CORBA servers using S objects and functions. They do not need to fuss with any of the details of using CORBA with C/C++.

In addition to being able to share functionality from other applications, the **CORBA** package provides a simple, high-level way to implement efficient and structured (i.e. fault tolerant and synchronized) parallel/distributed computing. All one need do is have different R processes running on different machines, each of which provides a CORBA server to perform part of bigger task. Then one R process acting as the manager calls these methods in the different servers as background or asynchronous tasks which run concurrently.

In addition to these interfaces, we are also working on embedding R in the Postgres database management system, the Apache Web Server and within Netscape.

The S language

The S language, originally developed at Bell Labs, has become a widely used medium for doing data analysis and for implementing the results of statistics research. Its influence was recognized in the 1998 ACM Software System award.

What about the future of S? What steps can we take that will increase its usefulness and relieve some of the current difficulties?

Compatibility between R and S-Plus

We regard as a major goal to define and support a growing compatibility at the level of an Application Programmer Interface between R and S-Plus. Much of the software described in this article can run in both R and S-Plus. This means that users of these and other new packages can take advantage of other facilities in both languages, or work on platforms that are only supported by one of the implementations.

There are many ways to make new software available in both implementations of the S language. Other things being equal, one would like the differences in the implementation to be minimized. To do this, one needs software and guidelines that implement a subset of the S language compatibly for both systems. As time goes by, we want that subset to grow.

A natural side-effect of such an approach is a possible "standard" for the S language. An agreement among users and those responsible for S-Plus and R on a standard application programming interface for the S language would enhance the value of the language. Potential users and programmers would be encouraged that software written against such a standard would be widely usable, just as such assurances are important for those programming in languages, such as C and C++, for which an agreed standard is available. Ongoing work in the Omegahat project will, we hope, encourage discussion of such a standard.

Summary

We have described very briefly some ongoing work, part of the Omegahat software, that provides facilities to programmers in R and S-Plus. Using these, you can connect software in the S language in an effective way to many other computing tools. Wherever possible, the resulting software should be directly usable in both R and S-Plus.

We encourage readers to get more information from the Omegahat website (http://www.omegahat.org/), and to try out the components that seem interesting for your applications. Because Omegahat is a joint, open-source project, we also particularly encourage suggestions and contributions to any of these efforts.

John M. Chambers
Bell Labs, Murray Hill, NJ, U.S.A.
jmc@research.bell-labs.com

Duncan Temple Lang
Bell Labs, Murray Hill, NJ, U.S.A.
duncan@research.bell-labs.com

Using XML for Statistics: The XML Package

by Duncan Temple Lang

XML, the eXtensible Markup Language is fast becoming the *next big thing* in computer applications and the Web. Along with the usual hype, there really is some substance and XML will probably turn out to have non-trivial relevance for statistics. In anticipation of this, we in the Omegahat project (http://www.omegahat.org/) have added an S package (i.e., works with both R and S-Plus) to both read and write XML. In this short article, we will outline some of the problems for which XML is a natural solution, and then describe what XML is, and some other areas in which XML technology is being exploited. Then we will take a brief look at some of the aspects of the R/S-Plus **XML** package. More information on the functions can be found at http://www.omegahat.org/RSXML/.

Using XML for datasets

Many of us are familiar with receiving the contents or values of a dataset in one file and a description of the dataset and its format in another file. Another problem is when the dataset is given as a set of files where records in one file correspond to records in the other files. In either case, the user must figure out the exact format and appropriate commands to group the values so as to read the values into into the data analysis environment. For example, the user must know whether some of the (initial) lines are comments; whether the first row of data contains variable names or is an actual record; how are missing values represented; whether integer values in a column represent a factor, a real valued-variable, or actually an integer; etc. The separation of the auxiliary information such as the ones just listed and other details such the author, the version of the dataset, the contact address, etc. can make it harder to process and ensure that the data has been read correctly.

When the datasets are not simply tables of numbers or strings, things quickly get more complicated. For example, if each record contains a different number of observations for a particular variable (i.e., ragged rows), some representation must be encoded into the ASCII file indicating which values are associated with which variable. Similarly, if the data frame of interest contains actual S objects that are not merely simple scalars, but are made up of a collection of values (e.g., the first two values of a record define a time interval object), we need to process these in a separate step after reading all the values using `read.table`.

If we could provide structure and additional information to data, we could also combine different, related datasets. For example, we can have both data about computers in a network and also the topology of that network in different parts of the same file. Also, we could add auxiliary information such as the number of records, the number of elements in a list, etc. This makes it easier and more efficient to read into S.

In order to address these issues, we need to find a way to add this type of structural information to the values. To do this, we need to agree on a format and then develop tools to read and write data using this format. And we need to develop these tools for different systems such as R, S-Plus, Matlab, SAS, etc. Well this seems like a lot of work, so we should look for an existing format that allows us to add our own types of information and which has existing tools to read and write data in this format. And this is exactly what XML is good for.

What is XML?

XML stands for eXtensible Markup Language. It is a general version of the familiar HTML. XML is "eXtensible" in the following way. While HTML has a fixed set of elements such as `<H1>`, `<H2>`, `<TABLE>`, `<TR>`, `<A>`, etc., XML allows one to define new tags or elements and the allowable attributes they support. In its simplest terms, XML is a general way of describing hierarchical data, i.e., trees. Nodes are represented as names with named attributes and subnodes, i.e.

```
<nodeName name="value" name="value">
  <subNode>...</subNode>
</nodeName>
```

How can we represent a dataset as a tree using XML? Simply by having the different elements such as row and variable names, the different records and the values within each record be introduced by XML tags. For example, we might markup the Motor Trend Car Data (`mtcars`) dataset in the following way:

```
<dataset numRecords="32">
  <variables number="11">
    <variable id="mpg">
      Miles Per U.S. Gallon
    </variable>
    <variable id="cyl">
      Number of cylinders
    </variable>
    ..
  </variables>
  <records>
    <record>
      <real>21</real>
      <int>6</int>
      <na/>...
    </record>
    ...
  </records>
</dataset>
```

With the ability to introduce new elements such as `<dataset>`, `<variable>`, etc., it is essential that software can interpret this so that it can convert this specification of a dataset to, for example, an S data frame. For this, S must expect certain tags and attributes an interpret them appropriately to create a dataset. We define these tags and the relationships between them in what is called a DTD—Document Type Definition. This is like a LaTeX style sheet, and characterizes a class of documents with the same structure. It specifies how the elements of a document are related and what are the valid attributes for each element.

Another difference between HTML and XML is that HTML is concerned with how things appear on a page or in a browser, whereas XML deals with the structure of data and the relationships between the different nodes. However, we often want to display XML documents such as the dataset above and have it be typeset in a nice way. This is where XSL, the eXtensible Stylesheet Language, enters the picture. XSL allows one to say specifies rules which control how XML elements are transformed to other XML elements, including HTML, and indirectly to TeX, PDF (via FOP), etc.

This separation of structure from appearance and the ability to easily transform XML documents to nicely rendered HTML, TeX, PDF, etc. files is one reason why XML is so useful. It allows us to put in real information about data that can be easily accessed from software. There are other reasons why XML is likely to be successful. The fact that is reasonably simple and builds on people's familiarity with HTML makes it likely that people will be able to use it. Since there is no company that controls the specification or has a singularly vested interest in controlling how it evolves, it has a good chance of being accepted widely and becoming a generic standard. Perhaps the most important reason that XML may succeed is the existence of a vast collection of applications and libraries for most common programming languages to parse XML, generate and edit XML, and convert XML to other formats using XSL. The support for different languages means that data in XML is essentially application independent and can be shared across different systems. Finally, XML is not a new technology. It is a simplification of SGML (Structured Generalized Markup Language) that has been in existence for about 20 years. Thus, XML has benefited from years of experience.

Other uses

We have seen how statisticians can use XML to share datasets across different applications. We can also use XML as an alternative format for saving R sessions and S objects in general. In other words, when S objects are written to disk, they can be stored using an XML format. This provides a convenient mechanism for sharing objects between R and S-Plus via a single copy of the object, reducing both the storage space and the work to convert them. Many types of these S language objects could also be read into XLisp-Stat, Octave and Matlab from their XML representation with little effort.

While we can use XML for our own purposes, it is important to think of XML as a way to exchange data with other disciplines too. XML is becoming very widely used for a variety of different applications. For example, many of the Microsoft applications use XML as one of their storage formats. Similarly, the

Gnumeric spreadsheet and other tools create output using XML. Businesses are using it to store information such as catalogs, documents, forms, etc. Many database systems now produce XML output as the result of queries. Also, business-to-business transactions are frequently defined by XML messages. And the Simple Object Access Protocol (SOAP, http://www.develop.com/soap/) is a way to use XML for distributed computing.

In addition to these more context specific uses of XML, there are several general purpose document type specifications (DTDs) that will prove to be relevant for statistics. These include:

MathML Mathematical formulae and expressions can be specified using MathML. The major browsers are adding facilities for rendering these within HTML documents. We might use MathML to represent formulas and mathematical functions which can be used in model fitting and trellis plots; contained in documentation; and passed to symbolic math environments such as Mathematica and Maple which can read and write MathML.

SVG The Scalable Vector Graphics (http://www.w3.org/Graphics/SVG/Overview.htm8) DTD provides a way to use XML to specify graphics, i.e., drawing, text and images. We might use this for describing plots generated in S, or rendering plots in S that were created in other systems. SVG supports dynamic and interactive plots, and even animation. There are already plug-ins for browsers that render SVG content.

GML The Geography markup language (GML) is used for representing "geographic information, including both the geometry and properties of geographic features". This includes maps, for example.

Bioinformatics Bioinformatic Sequence Markup Language (BSML) and a variety of other markup languages have been proposed for different aspects of genetic data.

There is a staggeringly large number of other domain-specific uses of XML listed at OASIS that may be worth browsing to see how people are using XML (http://www.oasis-open.org/cover/siteIndex.html#toc-applications).

XML schema is a new topic that will be interesting for us. It allows us to specify not only the structure of a document, but also give information about the types of different XML elements (e.g., `<real>` is a real number, `<int>` is an integer, etc.). Essentially, it allows type specification. This would allow us to read XML documents more efficiently and even to automatically generate C code to read datasets into S.

We have also started using XML for documenting S functions and datasets. This gives us an easier format to manipulate than the '.Rd' files so that we can add enhanced functionality to the documentation system (e.g., multiple, separate examples; output from commands; test code and output; etc.) in a more structured way. And we are working on ways to use XML for generating reports or statistical analysis output which are "live" documents containing text, code, output and plots that can be re-run with different datasets, etc.

The XML package

Now that we have discussed what XML is and what it can be used for, we should describe the basics of how to use the **XML** package in R. As we mentioned, XML is basically a way of representing trees. So, the basic functions of the **XML** package provide ways to read XML documents into R as a list of lists (i.e., a tree) and then access and manipulate the nodes in these trees using the familiar `[` and `[[` S operators.

The primary function of the package for reading XML documents is `xmlTreeParse` which returns the tree of *XMLNode* objects. Because we use Dan Veillard's C-level `libxml` parser, the XML can be read from a regular file, a compressed file, a URL or directly from a string. For example,

```
doc <-
  xmlTreeParse(paste("http://www.omegahat.org/",
                     "RSXML/examples/prompt.xml",
                     sep = ""))
doc <-
  xmlTreeParse(system.file("data", "mtcars.xml"))
```

The result contains not just the tree of *XMLNode* objects, but also information from the document's DTD. We can extract the different elements of tree by tag name or index. For example, we can get the first node

```
root <- xmlRoot(doc)
root[[1]]
```

which gives the collection of 11 `<variable>` nodes. We can get the first record either as

```
root[[2]]
```

or

```
root[["record"]]
```

We will usually want to do something with the tree and convert its contents to another form. This can be done after the call to `xmlTreeParse` and the tree has been created. Alternatively, we can provide one or more functions to `xmlTreeParse` which act as "handlers" for the different XML elements. These can process the *XMLNode* object associated with that XML element as it is encountered in the parsing and the function can return any object it wants to be put into the tree (including `NULL` to remove it from the tree). The use of handlers allows us to build quite simple filters for extracting data. For example, suppose we have measurements of height taken at monthly intervals for different subjects in a study, but that potentially different numbers of measurements were taken for each subject. Part of the dataset might look like

```
<record><id>193</id>
  <height unit="cm">
    <real>140.5</real>
    <real>143.4</real>
    <real>144.8</real>
  </height>
</record>
<record>
  <id>200</id>
  <height>
    <real>138.4</real>
  </height>
</record>
```

Now, suppose we want to get the records for the subjects which have more than one observation. We do this by specifying a handler for the `<record>` tags and counting the number of sub-nodes of the `<height>` element.

```
xmlTreeParse("data",
  handlers=list(record=function(x,...)
    ifelse(xmlSize(x[["height"]]) > 1, x, NULL)))
```

This will discard the second record in our example above, but keep the first. We should note we can also use XSL to filter these documents before reading the results of the filtering into S. More sophisticated things can be done in S, but XSL can be simpler at times and avoids a dependency on S. The good thing is that we have the luxury of choosing which approach to use.

The package also provides other facilities for reading XML documents using what is called event or SAX parsing which is useful for very large documents. Also, users can create XML documents or trees within S. They can be written to a file or a string by incrementally adding XML tags and content to different types of "connections".

The future

XML provides an opportunity to provide a universal, easy-to-use and standard format for exchanging data of all types. This has been a very brief introduction to some of the ways that we as statisticians may use XML. There is little doubt that there are many more. Also, regardless of how we can exploit XML for our own purposes, we will undoubtedly encounter it more as we interact with other disciplines and will have to deal with it in our daily work. Therefore, we should help to define formats for representing data types with which we work frequently and integrate the XML tools with ours.

Resources

The surge in popularity of XML has rivalled that of Java and hence the number of books on all aspects of XML has increased dramatically over the past few months. There are books on XML, XSL, XML with Java, etc. As with any dynamic and emerging technology, and especially one with so many subdomains with special interests, the Web is probably the best place to find information. The following are some general links that may help you find out more about XML:

- W3's XML page (http://www.w3.org/XML/)
- OASIS (http://www.oasis-open.org/)
- Apache's XML page (http://xml.apache.org/)

Duncan Temple Lang
Bell Labs, Murray Hill, NJ, U.S.A.
duncan@research.bell-labs.com

Programmer's Niche

by Bill Venables

Welcome and invitation

Welcome to the first installment of *Programmer's Niche*.

This is intended to be the first of a regular series of columns discussing R issues of particular interest to programmers. The original name was to have been *Programming Pearls* with the idea of specialising in short, incisive pearls of R programming wisdom, but we have decided to broaden it to include any matters of interest to programmers. The gems are still included, though, so if you have any favourites, let's hear about them.

That brings me to the real purpose of this preamble: to invite readers to contribute. I have offered to edit this column and occasionally to present an article but for the column to serve its purpose properly the bulk of the contributions must come from readers. If you have a particularly fine example of R programming and you can isolate the message succinctly, then let's hear about it. If you have a short expository idea for something you think is important and not well enough understood by other R programmers, then let's hear about that, too. On the other hand, if you have a question about R programming that has you stumped and you are just dying to get the views of the experts—then do send it along to `r-help`, there's a sport.

Please send contributions directly to me at `Bill.Venables@cmis.csiro.au`.

R profiling

One of the small miracles to come with R 1.2.0, at least for Unix and Linux platforms, is the ability to *profile* R functions. If you are a serious R programmer this is a tool you will come to use often and if you don't, you should.

The facility is expounded with an example in *"Writing R Extensions"* and I will not repeat it all here. Rather I will try to illustrate the idea with another, smaller example that I will take a bit further down the track.

Well, what is a profile? Any computation in R will involve calling at least one function. Under normal circumstances that function will itself call other functions and so on. The profile of the computation is a split up of the total time taken into components detailing how much time was spent inside each function called. Note that if function A calls function B then the time spent inside A will include the time spent inside B so under this interpretation the individual time components will not add up to the total time. It might be more useful to know the time spent in A excluding time spent in functions that are called from within A, so that the components do add to the total. In fact the profile provides both: the first is called the "total time" for function A and the second the "self time".

A convenient example is the well-known `subsets` function first used in MASS to illustrate the idea of a recursive function. This function finds all distinct subsets of size r from a set of size n. The set is defined by a vector, v. The results are presented as the rows of an $\binom{n}{r} \times r$ matrix. Here is a version of the function that has just a little more bulletproofing than the original version:

```
subsets0 <- function(n, r, v = 1:n) {
  if(r < 0 || r > n)
    stop("invalid r for this n")
  if(r == 0) vector(mode(v), 0) else
  if(r == n) matrix(v[1:n], 1, n) else
  rbind(cbind(v[1],
              Recall(n-1, r-1, v[-1])),
              Recall(n-1, r,   v[-1]))
}
```

The computation seems to come from nowhere but it is based on the simple premise that the subsets of size r are of two distinct types, namely those which contain `v[1]` and those that do not. This provides the essential divide-and-conquer step to allow recursion to take over and do the rest.

OK, let's give it a reasonably hefty example to chew up a bit of time and then use profiling to see how it has been spent. Before we do that, though, we need to make two small preliminary points.

Firstly note that not only is profiling at the moment restricted to Unix and Linux (because the current code uses Unix system calls), but to have it available at all you must install R on your machine with profiling enabled (which is the default); *"Writing R Extensions"* gives the details.

Secondly profiling is done in two steps, one within the R session and one outside. Inside R you turn profiling on and off with calls to the `Rprof` inbuilt function, for example:

```
> Rprof("Rsubs0.out")
> X <- subsets0(20, 6, letters)
> Rprof(NULL)
```

The two calls to `Rprof` start and stop a process where the computation is inspected at regular intervals (by default every 20 milliseconds) and the names of the functions currently on the evaluation stack are dumped onto an output file, in this example 'Rsubs0.out'. For long calculations this file can become quite large, of course.

Outside R there is a Perl script that can be used to summarise this information by giving the "total" and "self" times, and percentages, for each function called. It is called through R as shown in Figure 5.

The information is actually given twice: sorted by total time and by self time. I have only shown the second form here.

What really surprised me about this is the amount of self time taken by `vector`, `mode` and the ancillaries they use such as `switch` and `typeof`. I had thought this would be entirely negligible.

Given that clue let's recast the function so that the call to `vector` is done once and the value held in an enclosing environment. All the recursive hard work will then be done by a very lean internally-defined function, `sub`, that lives in the same environment. A version is given in Figure 6.

We have in fact added a few frills ("Do you want `v` to define a true set, so that repeated items are to be removed, or are you not fussed about repeats?") and removed all time consuming bulletproofing from the inner function, `sub`. I will not present the full profiling output, but the header says it all, really:

```
Each sample represents 0.02 seconds.
Total run time: 25.96 seconds.
....
```

That's a reduction of about 14 seconds in 40, at essentially no cost. It comes from realising that what

```
$ R CMD Rprof Rsubs0.out

Each sample represents 0.02 seconds.
Total run time: 39.48 seconds.

Total seconds: time spent in function and callees.
Self seconds: time spent in function alone.
....
   %       self        %       total
 self     seconds    total    seconds    name
 17.73      7.00     99.39     39.24     "cbind"
 11.65      4.60    100.00     39.48     "rbind"
  9.68      3.82     99.75     39.38     "Recall"
  9.47      3.74    100.00     39.48     "subsets0"
  6.64      2.62      6.64      2.62     "-"
  6.03      2.38      9.17      3.62     "matrix"
  5.02      1.98     17.43      6.88     "mode"
  4.81      1.90      7.50      2.96     "switch"
  4.51      1.78     21.94      8.66     "vector"
  3.85      1.52      4.51      1.78     "is.expression"
  3.70      1.46     14.13      5.58     "|"
  3.14      1.24      3.14      1.24     "=="
  2.74      1.08      3.14      1.24     "as.vector"
  2.74      1.08      6.69      2.64     ">"
  2.68      1.06      2.68      1.06     "typeof"
  2.48      0.98      3.75      1.48     "<"
  2.43      0.96      2.43      0.96     "!="
  0.30      0.12      0.30      0.12     ":"
  0.20      0.08      0.20      0.08     "is.call"
  0.20      0.08      0.20      0.08     "is.name"
```

Figure 5: Output of R CMD RProf

```
subsets1 <- function(n, r, v = 1:n, set = TRUE) {
  if(r < 0 || r > n) stop("invalid r for this n")
  if(set) {
    v <- unique(sort(v))
    if (length(v) < n) stop("too few different elements")
  }
  v0 <- vector(mode(v), 0)
  sub <- function(n, r, v) {  ## Inner workhorse
    if(r == 0) v0 else
    if(r == n) matrix(v, 1, n) else
      rbind(cbind(v[1], Recall(n-1, r-1, v[-1])),
                        Recall(n-1, r,   v[-1]))
  }
  sub(n, r, v[1:n])
}
```

Figure 6: Definition of function subsets1

seemed to be a computation done only rarely is in fact done very frequently and chomps away at the time.

That clue clearly indicates that if we can stop the recursion at an even earlier stage before it gets to the null sets it may pay off even more handsomely. The sets of size 1 are trivial to enumerate so let's take advantage of that with one extra line:

```
...
    if(r == 0) v0 else
    if(r == 1) matrix(v, n, 1) else
    ## the extra line
    if(r == n) matrix(v, 1, n) else
...
```

Now according to profiling on my machine the time for the same computation drops to just 12.32 seconds, less than one-third the original.

The story does not end there, of course. It seemed to me you could really make this computation zing (at the expense of memory, but hey, this is the 21st century) if you found a way to cache and re-use partial results as you went along. I did find a way to do this in S but using frame 0 or frame 1. Then Doug Bates neatly ported it to R making very astute use of the R scoping rules, function closures and environments, but that is another story for another time.

Bill Venables
CSIRO Marine Labs, Cleveland, Qld, Australia
Bill.Venables@cmis.csiro.au

Writing Articles for R News

or how (not) to ask for Christmas presents.

by Friedrich Leisch

Preface

When I wrote the call for articles for this first edition of R News on the evening of December 20, 2000 on my home laptop I shortly thought about which formats to accept. The decision that the newsletter itself would be produced in LaTeX had long been made, in fact we almost never use something different for text processing. I do a lot of interdisciplinary research with people from the management sciences where MS Word is the predominant text processor and hence am often confronted with conversion between '.doc' and '.tex' files when writing joint papers.

If the text uses only trivial markup (section headers, ...), then conversion is not too hard, but everybody can easily learn the little LaTeX that is involved in that, see the examples below. However, once mathematical equations or figures are involved, I know of no conversion that does not need considerable manual fixing to get a decent result (and we have tried a lot of routes). I considered including some words on Word, but then I thought: "Well, the email goes to `r-devel`, people that may be used to writing '.Rd' files—the format of R help files which is not unlike LaTeX—probably everybody knows LaTeX anyway, let's simply see if anybody really wants to write in Word.", sent the email and went to bed. Bad mistake...

The next day I had meetings in the morning and my first chance to read email was in the afternoon. To keep it short: I had started one of the perhaps most emotional (and longest) threads on `r-devel` so far, and as there is no such thing as a universal *best* word processing paradigm, there was of course also no "winner" in the discussion. So all I could add to the pro-LaTeX arguments 18 hours after my first email is that this editorial decision had already been made. I want to use this article to apologize to all for not being more detailed in my email that started the thread and explain the decision.

As all of R, R News is a volunteer project. We have no staff to do the editing or layouting etc., hence the editors have to "outsource" as much as possible to you, the prospective authors. This will only work if all use the same format, because—as explained above—automatic conversion simply does not work in practice. I know that the majority of R developers use LaTeX, hence the decision was not too hard which paradigm to choose.

The structure of R News articles

Figure 7 shows parts of the source code for this article. LaTeX is a markup language like, e.g., HTML and mixes layout commands and contents in a single text file (which you can write in any editor you like, or even Word). Most commands start with a backslash, arguments to the commands are usually given in curly brackets. We first specify the title, subtitle and author of the article and then issue the command `\maketitle` to actually typeset it, update the table of contents and PDF bookmarks. The command `\section*{}` starts a new section (`\section*{}` starts sections without numbering them, `\section{}` without the star would add numbers), and then we can simply enter the main text, separating paragraphs by blank lines.

The command `\file{}` has been defined by us to typeset file names in a different font and enclose them in single quotes. Finally we specify the au-

```
\title{Submitting to R News}
\subtitle{or how (not) to ask for Christmas presents.}
\author{by Friedrich Leisch}

\maketitle

\section*{Preface}

When I wrote the call for articles for this first edition of R News
on the evening of December 20, 2000 on my home laptop I shortly
thought about which formats to accept. The decision that the
newsletter itself would be produced in \LaTeX{} had long been made, in
fact we almost never use something different for text processing. I do
a lot of interdisciplinary research with people from the management
sciences where MS Word is the predominant text processor and hence am
often confronted with conversion between \file{.doc} and \file{.tex}
files when writing joint papers.

If the text uses ...

\address{Friedrich Leisch\\
  Technische Universität Wien, Austria\\
\email{Friedrich.Leisch@ci.tuwien.ac.at}}
```

Figure 7: The LaTeX source code of this article

thor's affiliation using our commands `\address{}` and `\email{}`. The double backslash in the code breaks lines (by default LaTeX treats single newlines just as whitespace). That's it, no magic involved at all.

The document style file for R News is still under constant changes while we are busy layouting this first issue, we will publish it on R's homepage as soon as we think that the interface is stable enough. There will also be a more in-depth guide for using it, that is beyond the scope of this article.

Graphics

In principle, graphics are much easier to convert between different formats than complete documents, and there are a number of good (and free) conversion tools available. One notable exception is WMF, the *Windows Meta Format*, which is hard to deal with on Unix-type platforms like our Linux machines.

Most graphics in this newsletter will probably be produced using R itself. In this case, we strongly prefer EPS (Encapsulated Postscript) format, which can easily be produced using R's `dev.copy2eps()` function on all platforms.

Summary

People being literate in LaTeX can write articles very much like it were a stand-alone document. For the moment, simply use the '`twocolumn`' option for layouting. The 'Rnews.sty' file will be made available as soon as possible. For others we will publish example articles on the web, and as long as you do not need special layouting it should be an easy go. Of course, typesetting mathematical equations in LaTeX (where its real strength is) is something completely different …

Friedrich Leisch
Technische Universität Wien, Austria
`Friedrich.Leisch@ci.tuwien.ac.at`

Upcoming Events

by Kurt Hornik

DSC 2001

The second international workshop on 'Distributed Statistical Computing' (DSC 2001) will take place at the Technische Universität Wien in Vienna, Austria from 2001-03-15 to 2001-03-17. This workshop will deal with future directions in statistical computing, such as event-driven software, graphical user interfaces, large and varied databases, reusable components, high-level object-oriented interfaces, and data exchange using XML.

Particular emphasis will be given to the R and Omegahat (`http://www.omegahat.org/`) projects. DSC 2001 builds on the spirit and success of DSC 1999, which was seminal to the further development of R and Omegahat.

This should be an exciting meeting for everyone interested in statistical computing with R.

There will be a two-day extension focusing on the future of the R project. Most members of the R Core Development Team will participate in this extension, which is open to everyone else interested.

The conference home page at `http://www.ci.tuwien.ac.at/Conferences/DSC-2001/` gives more information.

Kurt Hornik
Technische Universität Wien, Austria
`Kurt.Hornik@ci.tuwien.ac.at`