R News

The Newsletter of the R Project | Volume 1/2, June 2001

Editorial

by Kurt Hornik and Friedrich Leisch

Welcome to the second issue of *R News*, the newsletter of the R project for statistical computing. First we would like to thank for all the positive feedback we got on the first volume, which encourages us very much to continue with this online magazine.

A lot has happened since the first volume appeared last January, especially the workshop "DSC 2001" comes to mind. More on this remarkable event can be found in the new regular column "Recent Events". During DSC 2001 the R Core Team was happy to announce Stefano Iacus (who created the Macintosh port of R) as a new member of R Core: Welcome aboard, Stefano!

We have started collecting an annotated R bibliography, see http://www.R-project.org/doc/bib/R-publications.html. It will list books, journal articles, ... that may be useful to the R user community (and are related to R or S). If you are aware of any publications that should be included in the bibliography, please send email to R-core@R-project.org.

To make citation easier we have registered an ISSN number for this newsletter: ISSN 1609-3631. Please use this number together with the home page of the newsletter (http://cran.R-project.org/doc/Rnews/) when citing articles from *R News*.

We are now planning to have 4 issues of *R News* per year, scheduled for March/April, June/July, September/October and December/January, respectively. This should fit nicely into the calendar of the (Austrian) academic year and allow the editors to do most of the production work for at least 3 issues during semester breaks or holidays (remember that currently, all of the R project including this newsletter is volunteer work). Hence, the next issue is due after (northern hemisphere) summer: all contributions are welcome!

R 1.3.0 has just been released, and its new features are described in "Changes in R". Alongside, several add-on packages from CRAN have been categorized as *recommended* and will be available in all binary distributions of R in the future ("Changes on CRAN" has more details). *R News* will provide introductory articles on each recommended package, starting in this issue with "mgcv: GAMs and Generalized Ridge Regression for R".

Kurt Hornik
Wirtschaftsuniversität Wien, Austria
Technische Universität Wien, Austria
Kurt.Hornik@R-project.org

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

Contents of this issue:

Changes in R

by the R Core Team

New features in version 1.3.0

- Changes to connections:
  - New function `url()` to read from URLs. `file()` will also accept URL specifications, as will all the functions which use it.
  - File connections can now be opened for both reading and writing.
  - Anonymous file connections (via `file()`) are now supported.
  - New function `gzfile()` to read from / write to compressed files.
  - New function `fifo()` for connections to / from fifos (on Unix).
  - Text input from file, pipe, fifo, gzfile and url connections can be read with a user-specified encoding.
  - New functions `readChar()` to read character strings with known lengths and no terminators, and `writeChar()` to write user-specified lengths from strings.
  - `sink()` now has a stack of output connections, following S4.
  - `sink()` can also be applied to the message stream, to capture error messages to a connection. Use carefully!
  - `seek()` has a new 'origin' argument.
  - New function `truncate()` to truncate a connection open for writing at the current position.
  - New function `socketConnection()` for socket connections.
  - The 'blocking' argument for file, fifo and socket connections is now operational.
- Changes to date/time classes and functions:
  - Date/time objects now all inherit from class `"POSIXt"`.
  - New function `difftime()` and corresponding class for date/time differences, and a `round()` method.
  - Subtraction and logical comparison of objects from different date/time classes is now supported. NB: the format for the difference of two objects of the same date/time class has changed, but only for objects generated by this version, not those generated by earlier ones.
  - Methods for `cut()`, `seq()`, `round()` and `trunc()` for date/time classes.
  - Convenience generic functions `julian()`, `weekdays()`, `months()`, and `quarters()` with methods for `"POSIXt"` objects.
- Coercion from real to integer now gives `NA` for out-of-range values, rather than the most extreme integer of the same sign.
- The Ansari-Bradley, Bartlett, Fligner-Killeen, Friedman, Kruskal-Wallis, Mood, Quade, *t*, and Wilcoxon tests as well as `var.test()` in package **ctest** now have formula interfaces.
- The matrix multiplication functions `%*%` and `crossprod()` now use a level-3 BLAS routine dgemm. When R is linked with the ATLAS or other enhanced BLAS libraries this can be substantially faster than the previous code.
- New functions `La.eigen()` and `La.svd()` for eigenvector and singular value decompositions, based on LAPACK. These are preferred to `eigen()` and `svd()` for new projects and can make use of enhanced BLAS routines such as ATLAS. They are used in `cancor()`, `cmdscale()`, `factanal()` and `princomp()` and this may lead to sign reversals in some of the output of those functions.
- Provided the FORTRAN compiler can handle `COMPLEX*16`, the following routines now handle complex arguments, based on LAPACK code: `qr()`, `qr.coef()`, `qr.solve()`, `qr.qy()`, `qr.qty()`, `solve.default()`, `svd()`, `La.svd()`.
- `aperm()` uses strides in the internal C code and so is substantially faster (by Jonathan Rougier).
- The four `bessel[IJKY](x,nu)` functions are now defined for nu < 0.
- `[dpqr]nbinom()` also accept an alternative parametrization via the mean and the dispersion parameter (thanks to Ben Bolker).
- New generalised "birthday paradox" functions `[pq]birthday()`.
- `boxplot()` and `bxp()` have a new argument 'at'.
- New function `capabilities()` to report optional capabilities such as jpeg, png, tcltk, gzfile and url support.
- New function `checkFF()` for checking foreign function calls.

- New function `col2rgb()` for color conversion of names, hex, or integer.

- `coplot()` has a new argument 'bar.bg' (color of conditioning bars), gives nicer plots when the conditioners are factors, and allows factors for x and y (treated almost as if `unclass()`ed) using new argument 'axlabels'. [Original ideas by Thomas Baummann]

- 'hessian' argument added to `deriv()` and its methods. A new function `deriv3()` provides identical capabilities to `deriv()` except that 'hessian' is `TRUE` by default. `deriv(*, *, func = TRUE)` for convenience.

- New `dev.interactive()` function, useful for setting defaults for `par(ask=*)` in multifigure plots.

- `dist()` in package **mva** can now handle missing values, and zeroes in the Canberra distance.

- The default method for `download.file()` (and functions which use it such as `update.packages()`) is now `"internal"`, and uses code compiled into R.

- `eigen()` tests for symmetry with a numerical tolerance.

- New function `formatDL()` for formatting description lists.

- New argument 'nsmall' to `format.default()`, for S-PLUS compatibility (and used in various packages).

- `?/help()` now advertises `help.search()` if it fails to find a topic.

- `image()` is now a generic function.

- New function `integrate()` with S-compatible call.

- New function `is.unsorted()` the C version of which also speeds up `.Internal(sort())` for sorted input.

- `is.loaded()` accepts an argument 'PACKAGE' to search within a specific DLL/shared library.

- Exact p-values are available for the two-sided two-sample Kolmogorov-Smirnov test.

- `lm()` now passes '...' to the low level functions for regression fitting.

- Generic functions `logLik()` and `AIC()` moved from packages **nls** and **nlme** to **base**, as well as their lm methods.

- New components in `.Machine` give the sizes of `long`, `long long` and `long double` C types (or 0 if they do not exist).

- `merge.data.frame()` has new arguments, 'all[.xy]' and 'suffixes', for S compatibility.

- `model.frame()` now calls na.action with the terms attribute set on the data frame (needed to distiguish the response, for example).

- New generic functions `naprint()`, `naresid()`, and `napredict()` (formerly in packages **MASS** and **survival5**, also used in package **rpart**). Also `na.exclude()`, a variant on `na.omit()` that is handled differently by `naresid()` and `napredict()`.

  The default, lm and glm methods for `fitted()`, `residuals()`, `predict()` and `weights()` make use of these.

- New function `oneway.test()` in package **ctest** for testing for equal means in a one-way layout, assuming normality but not necessarily equal variances.

- `options(error)` accepts a function, as an alternative to an expression. (The Blue Book only allows a function; current S-PLUS a function or an expression.)

- `outer()` has a speed-up in the default case of a matrix outer product (by Jonathan Rougier).

- `package.skeleton()` helps with creating new packages.

- New `pdf()` graphics driver.

- `persp()` is now a generic function.

- `plot.acf()` makes better use of white space for 'nser > 2', has new optional arguments and uses a much better layout when more than one page of plots is produced.

- `plot.mts()` has a new argument 'panel' providing the same functionality as in `coplot()`.

- `postscript()` allows user-specified encoding, with encoding files supplied for Windows, Mac, Unicode and various others, and with an appropriate platform-specific default.

- `print.htest()` can now handle test names that are longer than one line.

- `prompt()` improved for data sets, particularly non-dataframes.

- `qqnorm()` is now a generic function.

- `read.fwf()` has a new argument 'n' for specifying the number of records (lines) read in.

- `read.table()` now uses a single pass through the dataset.

- `rep()` now handles lists (as generic vectors).
- `scan()` has a new argument 'multi.line' for S compatibility, but the default remains the opposite of S (records can cross line boundaries by default).
- `sort(x)` now produces an error when `x` is not atomic instead of just returning `x`.
- `split()` now allows splitting on a list of factors in which case their interaction defines the grouping.
- `stl()` has more optional arguments for fine tuning, a `summary()` and an improved `plot()` method.
- New function `strwrap()` for formatting character strings into paragraphs.
- New replacement functions `substr<-()` and `substring<-()`.
- Dataset `swiss` now has row names.
- Arguments 'pkg' and 'lib' of `system.file()` have been renamed to 'package' and 'lib.loc', respectively, to be consistent with related functions. The old names are deprecated. Argument 'package' must now specify a single package.
- The Wilcoxon and Ansari-Bradley tests now return point estimators of the location or scale parameter of interest along with confidence intervals for these.
- New function `write.dcf()` for writing data in Debian Control File format. `parse.dcf()` has been replaced by (much faster) internal `read.dcf()`.
- Contingency tables created by `xtabs()` or `table()` now have a `summary()` method.
- Functions `httpclient()`, `read.table.url()`, `scan.url()` and `source.url()` are now deprecated, and hence 'method="socket"' in `download.file()` is. Use url connections instead: in particular URLs can be specified for `read.table()`, `scan()` and `source()`.
- Formerly deprecated function `getenv()` is now defunct.
- Support for package-specific demo scripts (R code). `demo()` now has new arguments to specify the location of demos and to allow for running base demos as part of 'make check'.
- If not explicitly given a library tree to install to or remove from, respectively, `R CMD INSTALL` and `R CMD REMOVE` now operate on the first directory given in `R_LIBS` if this is set and non-null, and the default library otherwise.
- `R CMD INSTALL` and `package.description()` fix some common problems of 'DESCRIPTION' files (blank lines, ...).
- The `INSTALL` command for package installation allows a '`--save`' option. Using it causes a binary image of the package contents to be created at install time and loaded when the package is attached. This saves time, but also uses a more standard way of source-ing the package. Packages that do more than just assign object definitions may need to install with '`--save`'. Putting a file 'install.R' in the package directory makes '`--save`' the default behavior. If that file is not empty, its contents should be R commands executed at the end of creating the image.

  There is also a new command line option '`--configure-vals`' for passing variables to the configure script of a package.

- `R CMD check` now also checks the keyword entries against the list of standard keywords, for code/documentation mismatches (this can be turned off by the command line option '`--no-codoc`'), and for sufficient file permissions (Unix only). There is a new check for the correct usage of `library.dynam()`.

  It also has a new command line option '`--use-gct`' to use '`gctorture(TRUE)`' when running R code.

- `R CMD Rd2dvi` has better support for producing reference manuals for packages and package bundles.
- configure now tests for the versions of jpeg ($\geq$ 6b), libpng ($\geq$ 1.0.5) and zlib ($\geq$ 1.1.3). It no longer checks for the CXML/DXML BLAS libraries on Alphas.
- Perl scripts now use `Cwd::cwd()` in place of `Cwd::getcwd()`, as `cwd()` can be much faster.
- 'R::Dcf.pm' can now also handle files with more than one record and checks (a little bit) for continuation lines without leading whitespace.
- New manual 'R Installation and Administration' with fuller details on the installation process: file 'INSTALL' is now a brief introduction referencing that manual.
- New keyword 'internal' which can be used to hide objects that are not part of the API from indices like the alphabetical lists in the HTML help system.

- Under Unix, shlib modules for add-on packages are now linked against R as a shared library ('libR') if this exists. (This allows for improved embedding of R into other applications.)

- New mechanism for explicitly registering native routines in a DLL/shared library accessible via `.C()`, `.Call()`, `.Fortran()` and `.External()`. This is potentially more robust than the existing dynamic lookup, since it checks the number of arguments, type of the routine.

- New mechanism allowing registration of C routines for converting R objects to C pointers in `.C()` calls. Useful for references to data in other languages and libraries (e.g. C and hdf5).

- The internal ftp/http access code maintains the event loop, so you can download whilst running tcltk or Rggobi, say. It can be hooked into package **XML** too.

New features in version 1.2.3

- Support for configuration and building the Unix version of R under Mac OS X. (The 'classic' Macintosh port is 'Carbonized' and also runs under that OS.)

- `dotchart()` and `stripchart()` become the preferred names for `dotplot()` and `stripplot()`, respectively. The old names are now deprecated.

- Functions in package **ctest** now consistently use `+/-Inf` rather than `NA` for one-sided confidence intervals.

New features in version 1.2.2

- The Macintosh port becomes a full member of the R family and its sources are incorporated as from this release. See 'src/macintosh/INSTALL' for how that port is built.

- The API header files and export files 'R.exp' are released under LGPL rather than GPL to allow dynamically loaded code to be distributed under licences other than GPL.

- `postscript()` and `xfig()` devices now make use of genuine Adobe afm files, and warn if characters are used in string width or height calculations that are not in the afm files.

- Configure now uses a much expanded search list for finding a FORTRAN 77 compiler, and no longer disallows wrapper scripts for this compiler.

- New Rd markup `\method{GENERIC}{CLASS}` for indicating the usage of methods.

- `print.ftable()` and `write.ftable()` now have a 'digits' argument.

- `undoc()` has a new 'lib.loc' argument, and its first argument is now called 'package'.

Changes on CRAN

by Kurt Hornik and Friedrich Leisch

CRAN packages

The following extension packages from 'src/contrib' were added since the last newsletter.

AnalyzeIO Functions for I/O of ANALYZE image format files. By Jonathan L Marchini.

CoCoAn Two functions to compute correspondence analysis and constrained correspondence analysis and to make the associated graphical representation. By Stephane Dray.

GLMMGibbs Generalised Linear Mixed Models are an extension of Generalised Linear Models to include non-independent responses. This package allows them to be fitted by including functions for declaring factors to be random effects, for fitting models and generic functions for examining the fits. By Jonathan Myles and David Clayton.

GeneSOM Clustering Genes using Self-Organizing Map. By Jun Yan.

Oarray Generalise the starting point of the array index, e.g. to allow `x[0, 0, 0]` to be the first element of `x`. By Jonathan Rougier.

PTAk A multiway method to decompose a tensor (array) of any order, as a generalisation of SVD also supporting non-identity metrics and penalisations. 2-way SVD with these extensions is also available. The package includes also some other multiway methods: PCAn (Tucker-n) and PARAFAC/CANDECOMP with these extensions. By Didier Leibovici.

RArcInfo This package uses the functions written by Daniel Morissette (danmo@videotron.ca) to read geographical information in Arc/Info V 7.x format to import the coverages into R variables. By Virgilio Gómez-Rubio.

RMySQL DataBase Interface and MySQL driver for R. By David A. James and Saikat DebRoy.

RandomFields Simulating and analysing random fields using various methods. By Martin Schlather.

SuppDists Ten distributions supplementing those built into R. Inverse Gauss, Kruskal-Wallis, Kendall's Tau, Friedman's chi squared, Spearman's rho, maximum F ratio, the Pearson product moment correlation coefficiant, Johnson distributions, normal scores and generalized hypergeometric distributions. In addition two random number generators of George Marsaglia are included. By Bob Wheeler.

adapt Adaptive quadrature in up to 20 dimensions. By Thomas Lumley.

blighty Function for drawing the coastline of the United Kingdom. By David Lucy.

bqtl QTL mapping toolkit for inbred crosses and recombinant inbred lines. Includes maximum likelihood and Bayesian tools. By Charles C. Berry.

cramer Provides R routine for the multivariate nonparametric Cramer-Test. By Carsten Franz.

event.chart The function `event.chart` creates an event chart on the current graphics device. S original by J. Lee, K. Hess and J. Dubin. R port by Laura Pontiggia.

geoR Functions to perform geostatistical analysis including model-based methods. By Paulo J. Ribeiro Jr and Peter J. Diggle.

gregmisc Misc Functions written/maintained by Gregory R. Warnes. By Gregory R. Warnes. Includes code provided by William Venables and Ben Bolker.

lgtdl A very simple implementation of a class for longitudinal data. See the corresponding paper in the DSC-2001 proceedings. By Robert Gentleman.

lokern Kernel regression smoothing with adaptive local or global plug-in bandwidth selection., By Eva Herrmann (FORTRAN 77 & S original); Packaged for R by Martin Maechler.

lpridge `lpridge` and `lpepa` provide local polynomial regression algorithms including local bandwidth use. By Burkhardt Seifert (S original); packaged for R by Martin Maechler.

maxstat Maximally selected rank and Gauss statistics with several p-value approximations. By Torsten Hothorn and Berthold Lausen.

meanscore The Mean Score method for missing and auxiliary covariate data is described in the paper by Reilly & Pepe in Biometrika 1995. This likelihood-based method allows an analysis using all available cases and hence will give more efficient estimates. The method is applicable to cohort or case-control designs. By Marie Reilly PhD and Agus Salim.

netCDF Read data from netCDF files. By Thomas Lumley, based on code by Steve Oncley and Gordon Maclean.

nlrq Nonlinear quantile regression routines. By Roger Koenker and Philippe Grosjean.

odesolve This package provides an interface for the ODE solver lsoda. ODEs are expressed as R functions. By R. Woodrow Setzer with fixups by Martin Maechler.

panel Functions and datasets for fitting models to Panel data. By Robert Gentleman.

permax Functions intended to facilitate certain basic analyses of DNA array data, especially with regard to comparing expression levels between two types of tissue. By Robert J. Gray.

pinktoe Converts S tree objects into HTML/perl files. These web files can be used to interactively traverse the tree on a web browser. This web based approach to traversing trees is especially useful for large trees or for trees where the text for each variable is verbose. By Guy Nason.

sma The package contains some simple functions for exploratory microarray analysis. By Sandrine Dudoit, Yee Hwa (Jean) Yang and Benjamin Milo Bolstad, with contributions from Natalie Thorne, Ingrid Lönnstedt, and Jessica Mar.

sna A range of tools for social network analysis, including node and graph-level indices, structural distance and covariance methods, structural equivalence detection, $p*$ modeling, and network visualization. By Carter T. Butts.

strucchange Testing on structural change in linear regression relationships. It features tests/methods from the generalized fluctuation test framework as well as from the F test (Chow test) framework. This includes methods to fit, plot and test fluctuation processes (e.g., CUSUM, MOSUM, recursive/moving estimates) and F statistics, respectively. Furthermore it is possible to monitor incoming data online. By Achim Zeileis, Friedrich Leisch, Bruce Hansen, Kurt Hornik, Christian Kleiber, and Andrea Peters.

twostage Functions for optimal design of two-stage-studies using the Mean Score method. By Marie Reilly PhD and Agus Salim.

CRAN mirrors the R packages from the Omegahat project in directory 'src/contrib/Omegahat'. The following are recent additions:

OOP OOP style classes and methods for R and S-Plus. Object references and class-based method definition are supported in the style of languages such as Java and C++. By John Chambers and Duncan Temple Lang.

RGnumeric A plugin for the Gnumeric spreadsheet that allows R functions to be called from cells within the sheet, automatic recalculation, etc. By Duncan Temple Lang.

RJavaDevice A graphics device for R that uses Java components and graphics APIs. By Duncan Temple Lang.

RSPython Allows Python programs to invoke S functions, methods, etc. and S code to call Python functionality. This uses a general foreign reference mechanism to avoid copying and translating object contents and definitions. By Duncan Temple Lang.

SLanguage Functions and C support utilities to support S language programming that can work in both R and S-Plus. By John Chambers.

SNetscape R running within Netscape for dynamic, statistical documents, with an interface to and from JavaScript and other plugins. By Duncan Temple Lang.

SXalan Process XML documents using XSL functions implemented in R and dynamically substituting output from R. By Duncan Temple Lang.

Slcc Parses C source code, allowing one to analyze and automatically generate interfaces from S to that code, including the table of S-accessible native symbols, parameter count and type information, S constructors from C objects, call graphs, etc. By Duncan Temple Lang.

Recommended packages

The number of packages on CRAN has grown over 100 recently, ranging from big toolboxes with a long history in the S community (like **nlme** or **survival**) to more specialized smaller packages (like the geostatistical packages described later in this volume of *R News*). To make orientation for users, package developers and providers of binary distributions easier we have decided to start categorizing the contributed packages on CRAN.

Up to now there have been 2 categories of packages: Packages with priority *base* that come directly with the sources of R (like **ctest**, **mva** or **ts**) and all the rest. As a start for finer granularity we have introduced the new priority *recommended* for the following packages: **KernSmooth**, **VR** (bundle of **MASS**, **class**, **nnet**, **spatial**), **boot**, **cluster**, **foreign**, **mgcv**, **nlme**, **rpart** and **survival**.

Criteria for priority *recommended* are:

- Actively maintained and good quality code that can be installed on all (major) platforms.
- Useful for a wider audience in the sense that users "would expect this functionality" from a general purpose statistics environment like R.
- Depend only on packages with priority *base* or *recommended*.

All binary distributions of R (that are available from CRAN) will at least include the recommended packages in the future. Hence, if you write a package that depends on a recommended package it should be no problem for users of your package to meet this requirement (as most likely the recommended packages are installed anyway). In some sense the "minimum typical R installation" is the base system plus the recommended packages.

A final note: This is of course **no judgement at all** about the quality of all the other packages on CRAN. It is just the (subjective) opinion of the R Core Team what a minimum R installation should include. The criterion we used most for compiling the list of recommended packages was whether we thought a package was useful for a wide audience.

MacOS and MacOS X

Starting from release 1.3.0 of R two different versions of R for Macintosh are available at CRAN in two separate folders: 'bin/macos' and 'bin/macosx'.

The 'bin/macos' folder contains a version intended to run on Macintosh machines running System 8.6 to 9.1 and which will run on MacOS X as a carbon application, thus natively. The folder has the same structure as it was for release 1.2.x. Now the base distribution is available as two versions:

'rm130.sit' contains the base R distribution without the recommended packages;

'rm130_FULL.sit' contains the base distribution along with all recommended packages.

It is possible at a later time to download the archive 'recommended.sit' that contains the additional recommended packages not included in the base-only package. As usual the other contributed packages can be found in a separate folder.

The 'bin/macosx' folder contains a MacOS X specific build that will run on a X11 server and it is based on the Darwin kernel, i.e., it is a Unix build that runs on MacOS X. This is provided by Jan de Leeuw (deleeuw@stat.ucla.edu). It comes in three versions:

'R-1.3.0-OSX-base.tar.gz' has the R base distribution. It has Tcl/Tk support, but no support for GNOME.

'R-1.3.0-OSX-recommended.tar.gz' has the R base distribution plus the recommended packages. It has Tcl/Tk support, but no support for GNOME.

'R-1.3.0-OSX-full.tar.gz' has the R base distribution plus 134 compiled packages. It is compiled with both GNOME and Tcl/Tk support.

The 'bin/macosx' folder contains two folders, one containing some additional dynamic libraries upon on which this port is based upon, and another giving replacements parts complied with the ATLAS optimized BLAS.

'ReadMe.txt' files are provided for both versions.

Other changes

GNU a2ps is a fairly versatile any-text-to-postscript processor, useful for typesetting source code from a wide variety of programming languages. 's.ssh', 'rd.ssh' and 'st.ssh' are a2ps style sheets for S code, Rd documentation format, and S transscripts, respectively. These will be included in the next a2ps release and are currently available from the "Other Software" page on CRAN.

Kurt Hornik
Wirtschaftsuniversität Wien, Austria
Technische Universität Wien, Austria
Kurt.Hornik@R-project.org

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

Date-Time Classes

by Brian D. Ripley and Kurt Hornik

Data in the form of date and/or times are common in some fields, for example times of diagnosis and death in survival analysis, trading days and times in financial time series, and dates of files. We had been considering for some time how best to handle such data in R, and it was the last of these examples that forced us to the decision to include classes for dates and times in R version 1.2.0, as part of the **base** package.

We were adding the function `file.info`. Finding information about files looks easy: Unix users take for granted listings like the following (abbreviated to fit the column width):

```
auk% ls -l
total 1189
...       948 Mar 20 14:12 AUTHORS
...      9737 Apr 24 06:44 BUGS
...     17992 Oct  7  1999 COPYING
...     26532 Feb  2 18:38 COPYING.LIB
...      4092 Feb  4 16:00 COPYRIGHTS
```

but there are a number of subtle issues that hopefully the operating system has taken care of. (The example was generated in the UK in April 2001 on a machine set to the C locale.)

- The format. Two formats are used in the extract above, one for files less than 6 months' old and one for older files. Date formats have an international standard (ISO 8601), and this is not it! In the ISO standard the first date is 2001-03-20 14.12. However, the format is not even that commonly used in the UK, which would be 20 Mar 2001 14:12. The month names indicate that this output was designed for an anglophone reader. In short, the format should depend on the *locale*.

- Time zones. Hopefully the times are in the time zone of the computer reading the files, and take daylight saving time into account, so the first time is in GMT and the second in BST. Somewhat more hopefully, this will be the case even if the files have been mounted from a machine on another continent.

  Note that this can be an issue even if one is only interested in survival times in days. Suppose a patient is diagnosed in New Zealand and dies during surgery in California?

We looked at existing solutions, the R packages **chron** and **date**. These seem designed for dates of the accuracy of a day (although **chron** allows partial days), are US-centric and do not take account of time zones. It was clear we had to look elsewhere for a comprehensive solution.

The most obvious solution was the operating system itself: after all it knew enough about dates to process the file dates in the example above. This was the route we decided to take.

Another idea we had was to look at what the database community uses. The SQL99 ISO standard (see Kline & Kline, 2001) has data types `date`, `time`, `time with time zone`, `timestamp`, `timestamp with time zone` and `interval`. The type `timestamp with time zone` looks to be what we wanted. Unfortunately, what is implemented in the common databases is quite different, and for example the MySQL data type `timestamp` is for dates after 1970-01-01 (according to Kline & Kline).

S-PLUS 5.x and 6.x have S4 classes `"timeDate"` and `"timeSpan"` for date-times and for time intervals. These store the data in whole days (since some origin for `"timeDate"`) and whole milliseconds past midnight, together with a timezone and a preferred printing format.

POSIX standards

We were aware that portability would be a problem with using OS facilities, but not aware how much of a headache it was going to be. The 1989 ISO C standard has some limited support for times, which are extended and made more precise in the ISO C99 standard. But there are few (if any) C99-conformant compilers. The one set of standards we did have a chance with was the POSIX set. Documentation on POSIX standards is hard to come by, Lewine (1991) being very useful if now rather old. Vendors do tend to comply with POSIX (at least superficially) as it is mandated for government computer purchases.

The basic POSIX measure of time, *calendar time*, is the number of seconds since the beginning of 1970, in the UTC timezone (GMT as described by the French). Even that needs a little more precision. There have been 22 *leap seconds* inserted since 1970 (see the R object `.leap.seconds`), and these should be discarded. Most machines would store the number as a signed 32-bit integer, which allows times from the early years of the 20th century up to 2037. We decided this was restrictive, and stored the number of seconds as a C double. In principle this allows the storage of times to sub-second accuracy, but that was an opportunity we overlooked. Note that there is little point in allowing a much wider range of times: timezones are a 19th century introduction, and countries changed to the Gregorian calendar at different dates (Britain and its colonies in September 1752). The corresponding R class we called `POSIXct`.

The raw measure of time is not easily digestible, and POSIX also provides a *'broken-down time'* in a `struct tm` structure. This gives the year, month, day of the month, hour, minute and second, all as integers. Those members completely define the clock time once the time zone is known [1]. The other members, the day of the week, the day of the year (0–365) and a DST flag, can in principle be calculated from the first six. Note that year has a baseline of 1900, so years before 1970 are clearly intended to be used. The corresponding R class we called `POSIXlt` (where the '`lt`' stands for "local time"), which is a list with components as integer vectors, and so can represent a vector of broken-down times. We wanted to keep track of timezones, so where known the timezone is given by an attribute `"tzone"`, the name of the timezone.

Conversions

A high-level language such as R should handle the conversion between classes automatically. For times within the range handled by the operating system we can use the POSIX functions `mktime` to go from broken-down time to calendar time, and `localtime` and `gmtime` to go from calendar time to broken-down time, in the local timezone and UTC respectively. The only way to do the conversion in an arbitrary timezone is to set the timezone *pro tem*[2]. That proved difficult!

We also want to be able to print out and scan in date-times. POSIX provides a function `strftime` to print a date-time in a wide range of formats. The reverse function, `strptime`, to convert a character string to a broken-down time, is not in POSIX but is widely implemented.

Let us look again at the file dates, now using R:

```
> file.info(dir())[, "mtime", drop=FALSE]
                              mtime
AUTHORS       2001-03-20 14:12:22
BUGS          2001-04-24 06:44:10
COPYING       1999-10-07 19:09:39
COPYING.LIB   2001-02-02 18:38:32
COPYRIGHTS    2001-02-04 16:00:49
...
```

This gives the dates in the default (ISO standard) format, and has taken proper account of the timezone change. (Note that this has been applied to a column of a data frame.) When printing just a date-time object the timezone is given, if known. We can easily use other formats as we like.

```
> zz <- file.info(dir())[1:5, "mtime"]
> zz
[1] "2001-03-20 14:12:22 GMT"
[2] "2001-04-24 06:44:10 BST"
[3] "1999-10-07 19:09:39 BST"
[4] "2001-02-02 18:38:32 GMT"
[5] "2001-02-04 16:00:49 GMT"
> format(zz, format="%x %X")
```

[1] ISO C99 adds members `tm_zone` and `tm_leapseconds` in `struct tmx`. It represents both the time zone and the DST by an offset in minutes, information that is not readily available in some of the platforms we looked at.

[2] C99 has functions `zonetime` and `mkxtime` which would avoid this.

```
# locale specific: see also %c or %C.
[1] "03/20/01 14:12:22" "04/24/01 06:44:10"
[3] "10/07/99 19:09:39" "02/02/01 18:38:32"
[5] "02/04/01 16:00:49"
> Sys.setlocale(locale = "en_UK")
[1] "en_UK"
> format(zz, format="%x %X")
[1] "20/03/01 02:12:22 PM" "24/04/01 06:44:10 AM"
[3] "07/10/99 07:09:39 PM" "02/02/01 06:38:32 PM"
[5] "04/02/01 04:00:49 PM"
> format(zz, format="%b %d %Y")
[1] "Mar 20 2001" "Apr 24 2001" "Oct 07 1999"
[4] "Feb 02 2001" "Feb 04 2001"
> format(zz, format="%a %d %b %Y %H:%M")
[1] "Tue 20 Mar 2001 14:12"
[2] "Tue 24 Apr 2001 06:44"
[3] "Thu 07 Oct 1999 19:09"
[4] "Fri 02 Feb 2001 18:38"
[5] "Sun 04 Feb 2001 16:00"
```

It was easy to add conversions from the `chron` and `dates` classes.

The implementation

The original implementation was written under Solaris, and went very smoothly. It was the only OS for which this was the case! Our idea was to use OS facilities where are these available, so we added simple versions of `mktime` and `gmtime` to convert times far into the past or the future ignoring timezones, and then worked out the adjustment on the same day in 2000 in the current timezone.

One advantage of an Open Source project is the ability to borrow from other projects, and we made use of `glibc`'s version of `strptime` to provide one for platforms which lacked it.

Coping with the vagaries of other platforms proved to take far longer. According to POSIX, `mktime` is supposed to return `-1` (which is a valid time) for out-of-range times, but on Windows it crashed for times before 1970-01-01. Such times were admittedly pre-Microsoft! Linux systems do not normally have a `TZ` environment variable set, and this causes crashes in `strftime` when asked to print the timezone, and also complications in temporarily setting timezones (there is no way portably to unset an environment variable from C). Some platforms were confused if the DST flag was set to `-1` ('unknown'). SGI's `strptime` only works after 1970. And so on .... The code became more and more complicated as workarounds were added.

We provided a configure test of whether leap seconds were ignored, and code to work around it if they are not. We never found such a platform, but we have since had a bug report which shows they do exist and we did not get the code quite right first time around.

Describing all the problems we found would make a very long article. We did consider providing all our own code based on `glibc`. In retrospect that would have saved a lot of problems, but created others. Managing a timezone database is really tedious, and we would have had to find out for each OS how to read the local timezone in terms that `glibc` would understand.

Much later we found out that classic MacOS does not really understand timezones, and so workarounds had to be added for that port of R.

Extensions

The implementation we put in version 1.2.0 was not fully complete. One issue which arose was the need to form time differences (the SQL99 `interval` data type). Subtraction of two `POSIXct` or two `POSIXlt` times gave a number of seconds, but subtracting a `POSIXct` time from a `POSIXlt` time failed.

Version 1.3.0 provides general facilities for handling time differences, via a class `"difftime"` with generator function `difftime`. This allows time units of days or hours or minutes or seconds, and aims to make a sensible choice automatically. To allow subtraction to work within R's method dispatch system we needed to introduce a super-class `"POSIXt"`, and a method function `-.POSIXt`. Thus from 1.3.0, calendar-time objects have class `c("POSIXt", "POSIXct")`, and broken-down-time objects have class `c("POSIXt", "POSIXlt")`. Appending the new class rather than prepending would not work, for reasons we leave as an exercise for the reader.

Here is an example of the time intervals between R releases:

```
> ISOdate(2001, 2, 26) - ISOdate(2001, 1, 15)
Time difference of 42 days
> ISOdate(2001, 4, 26) - ISOdate(2001, 2, 26)
Time difference of 59 days
```

The result is of class `"difftime"` and so printed as a number of days: it is stored as a number of seconds.

One has to be slightly careful: compare the second example with

```
> as.POSIXct("2001-04-26") -
    as.POSIXct("2001-02-26")
Time difference of 58.95833 days
> c(as.POSIXct("2001-04-26"),
    as.POSIXct("2001-02-26"))
[1] "2001-04-26 BST" "2001-02-26 GMT"
> c(ISOdate(2001, 4, 26), ISOdate(2001, 2, 26))
[1] "2001-04-26 13:00:00 BST"
[2] "2001-02-26 12:00:00 GMT"
```

The difference is that `ISOdate` chooses midday GMT as the unspecified time of day, and `as.POSIXct` is using midnight in the timezone. As the UK changed to DST between the releases, had the releases occurred at the same time of day the interval would not have been an exact multiple of a 24-hour day. The `round` method can be useful here.

There are many more things one would like to do with date-time objects. We want to know the current time (`Sys.time`) and timezone (`Sys.timezone`).

Methods for `format` provide very flexible ways to convert them to character strings. We have an `axis` method to use them to label graphs. Lots of methods are needed, for `all.equal`, `as.character`, `c`, `cut`, `mean`, `round`, `seq`, `str`, ... And these need to check for appropriateness, so for example sums of dates are not well-defined, whereas means are.

We have also provided convenience functions like `weekdays`, `months` and `quarters`, which either extract information from the `POSIXlt` list or convert using an appropriate `format` argument in a call to the `format` method. The `POSIXt` method for the (new) generic function `julian` converts to Julian dates (the number of days since some origin, often 1970-01-01).

The future

We believe that the date-time classes in base R now provide sufficient flexibility and facilities to cover almost all applications and hence that they should now be used in preference to earlier and more limited systems. Perhaps most important is that these classes be used in inter-system applications such as database connectivity.

References

International Organization for Standardization (1988, 1997, ...) ISO 8601. Data elements and interchange formats – Information interchange – Representation of dates and times. The 1997 version is available on-line at ftp://ftp.qsl.net/pub/g1smd/8601v03.pdf.

Kline, K. and Kline, D. (2001) *SQL in a Nutshell.* O'Reilly.

Lewine, D. (1991) *POSIX Programmer's Guide. Writing Portable UNIX Programs.* O'Reilly & Associates.

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

Kurt Hornik
Wirtschaftsuniversität Wien, Austria
Technische Universität Wien, Austria
Kurt.Hornik@R-project.org

Installing R under Windows

by Brian D. Ripley

Very few Windows users will have ever experienced compiling a large system, as binary installations of Windows software are universal. Further, users are used to installing software by a point-and-click interface with a minimum of reading of instructions, most often none. The expectation is

> Insert the CD.
>
> If it doesn't auto-run, double-click on a file called `Setup.exe` in the top directory.
>
> Go through a few 'Wizard' pages, then watch a progress bar as files are installed, then click on Finish.

Contrast this with '`untar` the sources, run `./configure`, `make` then `make install`'. Each in its own way is simple, but it is really horses for courses.

Every since Guido Masarotto put out a version of R for Windows as a set of `zip` files we have been looking for a way to install R in a style that experienced Windows users will find natural. At last we believe we have found one.

When I first looked at this a couple of years ago most packages (even Open Source ones) used a commercial installer such as InstallShield or Wise. Although I had a copy of InstallShield, I was put off by its size, complexity and the experiences I gleaned, notably from Fabrice Popineau with his `fptex` installation.

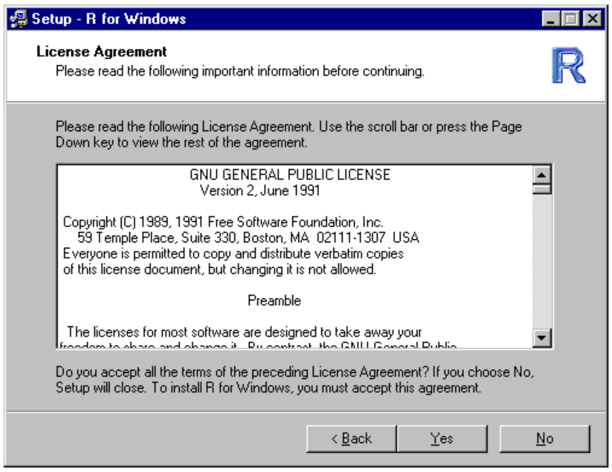
Shortly afterwards, MicroSoft introduced their own installer for their Office 2000 suite. This works in almost the same way, except that one double-clicks on a file with extension `.msi`. There is a development kit for this installer and I had expected it to become the installer of choice, but it seems rarely used. (The `Perl` and now `Tcl` ports to Windows do use it.) That makes one of its disadvantages serious: unless you have a current version of Windows (ME or 2000) you need to download the installer `InstMsi.exe`, which is around 1.5Mb and whose installation needs privileges an ordinary user may not have.

In May 1999 I decided to write a simple installer, `rwinst.exe`, using the GraphApp toolkit that Guido had used to write the R for Windows GUI, and this has been in use since. But it was not ideal, for

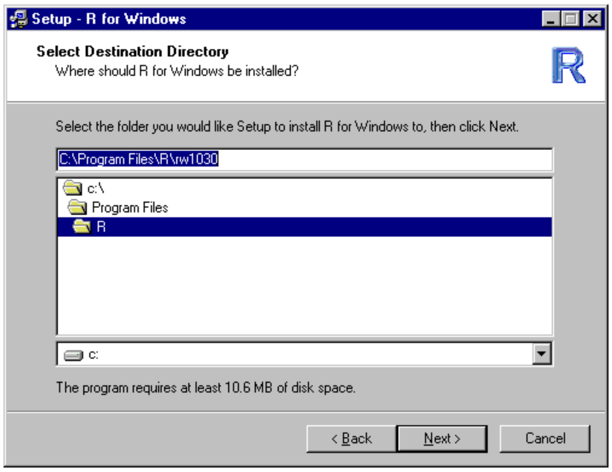
- It did not use a completely standard 'look-and-feel'.
- It was hard to maintain, and mistakes in an installer are 'mission-critical'.
- Users found it hard to cope with needing several files to do the installation.

The prospect of *recommended* packages at version 1.3.0 was going to require twice as many files, and forced a re-think in early 2001. I had earlier looked at Inno Setup by Jordan Russell (www.jrsoftware.org)

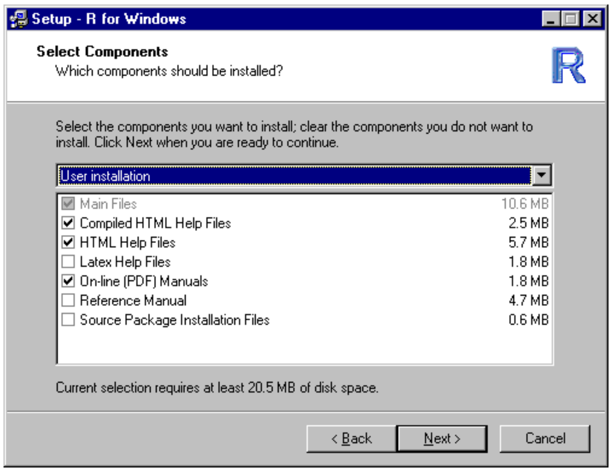
(a) The 'welcome' screen



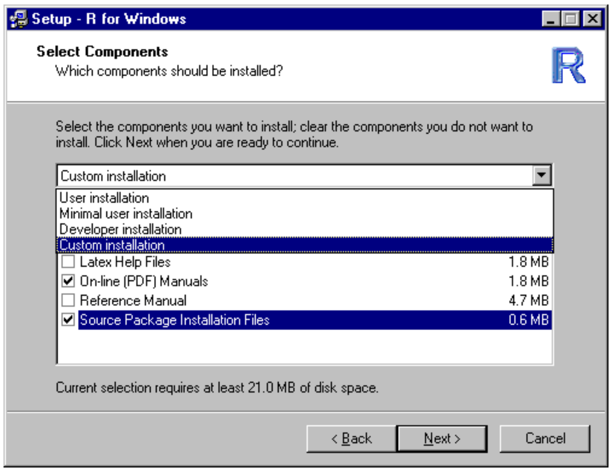
(b) The licence – GPL2 of course



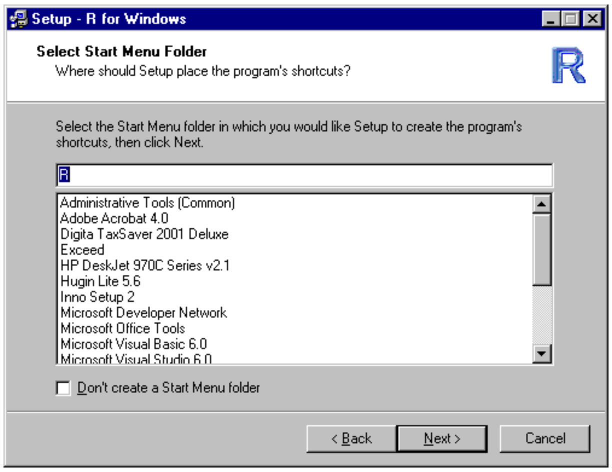
(c) Choose the installation folder



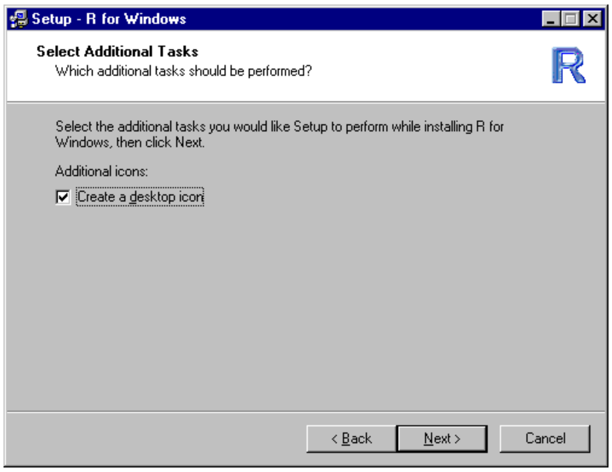
(d) Select components as required



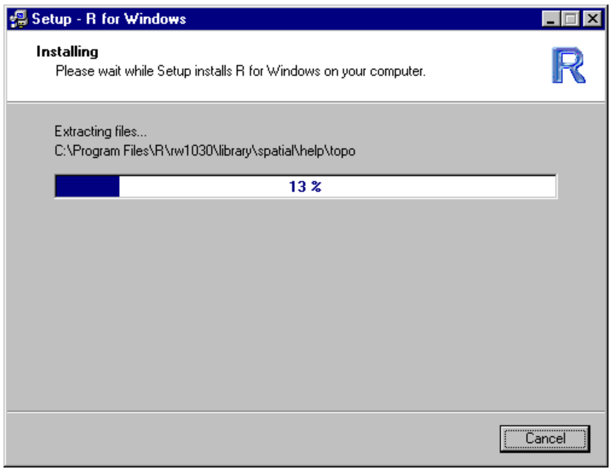
(e) There is a list of standard selections, plus 'custom'



(f) Choose a group name for the Start menu



(g) Do you want a desktop icon?



(h) Installing the files

and had found it too limited. However, an enhanced version 2 was by then in beta release and was used to make an alternative installer for R 1.2.3. Inno Setup 2 is now released and has been adopted for the recommended installers for R 1.3.0. The figures show the installation sequence using `SetupR.exe` for 1.3.0. Note that there is a fair degree of control but with sensible defaults. For example, the commercial installers do not usually offer the option **not** to add to the start menu, nor to add a desktop icon (and that is labelled by the R version number).

The `SetupR.exe` installation is a single file of about 15Mb, which can conveniently be distributed on CD-R or downloaded (over a fast enough Internet connection).

One issue that had been a concern was that it is thought that some users need to install R from floppies, and the `.zip` files had been designed where possible[1] to fit on a single floppy. How much demand there was/is is hard to gauge, and such users almost by definition are unlikely to be fairly represented by email contact. As R grew, maintaining a division into floppy-sized pieces became harder, and needed constant vigilance. Fortunately Inno Setup provided a simple option to produce (exactly) floppy-sized pieces, and we have an alternative distribution consisting of `miniR.exe` and `miniR-1.bin` which fit on one floppy, and `mini-2.bin` to `mini-6.bin` which fit on five further floppies. The six floppies are very full, and doubtless the next release will need seven.

Inno Setup is a Delphi program which is freely available (including source). It is programmed by a script of instructions. For a project the size of R that script needs to be a few thousand lines, but is generated automatically by a `Perl` script from the distribution files. I had a workable installer running in a couple of hours, have spent less than a day on it in total, and future maintenance of this installer should be a much easier task

Uninstalling

Surely you wouldn't want to remove something as useful as R?

One good reason to remove R is to clear space before upgrading to a new version. In the past the only way to do this was to delete the whole directory containing R, say `c:\R\rw1021`. That did remove everything, as R touched nothing else (it did not use the Registry nor the system directories), but it would also wipe out any customizations the user had made, as well as all the add-on packages installed in `c:\R\rw1021\library`.

Inno Setup automatically provides an uninstaller, and an R installation made with `setupR.exe` or `miniR.exe` can be removed via the R group on the `Start` menu, from the Control Panel[2] or by running `Uninst.exe` in the top directory of the installation. This does only uninstall the files it installed, so that added packages are left. It also removes the Registry entries it set.

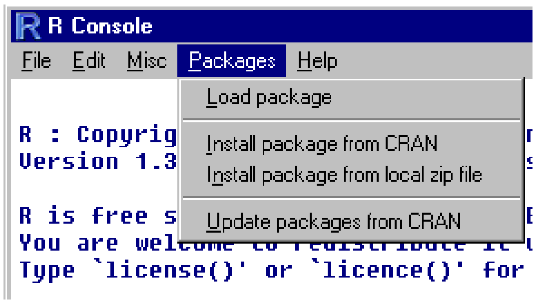
This makes upgrading easy. Just uninstall the old version, install the new one and move the added packages across to `c:\R\rw1030\library` (say). **Note:** some packages may need to be updated to work with new versions of R, so as a final step run `update.packages()`. Which brings us to ...

Installing packages

Windows' users have almost all expected to have packages pre-compiled for them, and to have a

[1] this was never possible for the `.zip` file containing the 4.6Mb PDF version of the reference manual.
[2] for sufficiently privileged users only

simple way to install them. Another function of `rwinst.exe` was to provide a point-and-click way to install pre-compiled packages. Now `rwinst.exe` is no longer supported[3], we needed another way to ease the installation of packages. This is provided by the Packages menu introduced in R 1.3.0. This takes advantage of the new facilities to download files from URLs to allow installation of packages either from a local zip file or directly from a CRAN node.



Other items on that menu provide shortcuts to `library()` and `update.packages()`.

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

Spatial Statistics in R

by Brian D. Ripley

The following two articles discuss two recent spatial statistics packages for R, and the Editors suggest that I write an overview of the area.

There are several packages available, almost all of which originated as S code and from quite different communities. As a result there is considerable overlap in their functionality. Many are quite old (pre-dating classes in S, for example, and from an era of much lower levels of computing resources). In roughly historical order there are

akima An R interface by Albrecht Gebhardt to spatial spline interpolation Fortran code by H. Akima. Closely modelled on the `interp` function in S-PLUS.

tripack Delaunay triangulation of spatial data. An R interface by Albrecht Gebhardt to Fortran code by R. J. Renka.

spatial Now part of the **VR** bundle. Contains trend-surface analysis, kriging and point-process code originally written by B. D. Ripley in 1990–1 for teaching support.

VR version 6.2-6 includes enhanced trend-surface code contributed by Roger Bivand.

sgeostat Geostatistical modelling code written for S by James J. Majure and ported to R by Albrecht Gebhardt.

splancs Originally commercial code for spatial and space-time point patterns by Barry Rowlingson. Roger Bivand has made a GPL-ed version available for R, with contributions from Giovanni Petris.

spatstat Code for point pattern analysis originally written for S-PLUS 5.1 by Adrian Baddeley and Rolf Turner. The version on the website (http://www.maths.uwa.edu.au/~adrian/spatstat.html) is said to work with R 1.0.1.

geoR Geostatistical code by Paulo J. Ribeiro. See the next article. Paulo does not mention that he also has a **geoS**.

RandomFields Specialized code to simulate from continuous random fields by Martin Schlather. See the next but one article.

Which should you use? Spatial statistics is not a single subject, and it depends which part you want. It is natural for me to use the classification of Ripley (1981).

Smoothing and interpolation is still a range of methods. Package **akima** provides one reasonable interpolation method, and package **spatial** provides the commonest of smoothing methods, trend surfaces. Kriging can be either interpolation or smoothing, and is covered at various depths in **spatial**, **sgeostat** and **geoR**.

For **spatial autocorrelation** there is nothing available yet. For spatial lattice process (e.g. CAR processes) there is no specialized code, but `gls` from **nlme** could be used.

For **spatial point patterns**, **spatial** provides the basic tools, and methods to fit Strauss processes. **splancs** handles polygonal boundaries, and it and **spatstat** have a wider range of methods (by no means all of which I would recommend).

[3]it can still be compiled from the sources.

It is worth mentioning the commercial module **S+SpatialStats** for S-PLUS, which covers all the areas mentioned here, and is **not** available for R. The prospect of such a module, and later of further development of it, has dampened enthusiasm for user-contributed spatial statistics code over much of the last decade. It is worth bearing in mind that a great deal of the current wealth of packages for R emanates from the work of users filling gaps they saw in S and its commercial offspring.

Bibliography

[1] B. D. Ripley. *Spatial Statistics*. Wiley, 1981.

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

geoR: A Package for Geostatistical Analysis

by Paulo J. Ribeiro Jr and Peter J. Diggle

geoR is a package to perform geostatistical data analysis and spatial prediction, expanding the set of currently available methods and tools for analysis of spatial data in R. It has been developed at the Department of Mathematics and Statistics, Lancaster University, UK. A web site with further information can be found at: http://www.maths.lancs.ac.uk/~ribeiro/geoR.html.

Preliminary versions have been available on the web for the last two years. Based on users' feedback and on our own experiences, we judge that the package has been used mainly to support teaching material and to carry out data analysis and simulation studies for scientific publications.

Package **geoR** differs from the other R tools for geostatistical data analysis in following the model-based inference methods described in (3).

Spatial statistics and geostatistics

Spatial statistics is the collection of statistical methods in which spatial locations play an explicit role in the analysis of data. The main aim of *geostatistics* is to model continuous spatial variation assuming a basic structure of the type $Y(x) : x \in \mathbb{R}^d, d = 1, 2$ or 3 for a random variable Y of interest over a region. Characteristic features of geostatistical problems are:

- data consist of *responses* Y_i associated with *locations* x_i which may be non-stochastic, specified by the sampling design (e.g. a lattice covering the observation region A), or stochastic but selected independently of the process $Y(x)$.
- in principle, Y could be determined from any location x within a continuous spatial region A.
- $\{Y(x) : x \in A\}$ is related to an unobserved stochastic process $\{S(x) : x \in A\}$, which we call the *signal*.
- scientific objectives include prediction of one or more functionals of the stochastic process $\{S(x) : x \in A\}$.

Geostatistics has its origins in problems connected with estimation of reserves in mineral exploration/mining (5). Its subsequent development, initially by Matheron and colleagues at École des Mines, Fontainebleau (8) was largely independent of "mainstream" spatial statistics. The term "kriging" was introduced to describe the resulting methods for spatial prediction. Earlier developments include work by Matérn (6, 7) and by Whittle (10). Ripley (9) re-casts kriging in the terminology of stochastic process prediction, and this was followed by significant cross-fertilisation during 1980's and 1990's (eg the *variogram* is now a standard statistical tool for analysing correlated data in space and/or time). However, there is still vigorous debate on practical issues such as how to perform inference and prediction, and the role of explicit probability models.

The Gaussian model

The currently available functions on **geoR** assume a basic model specified by:

1. a signal $S(\cdot)$ which is a stationary Gaussian process with
   (a) $\mathrm{E}[S(x)] = 0$,
   (b) $\mathrm{Var}\{S(x)\} = \sigma^2$,
   (c) $\mathrm{Corr}\{S(x), S(x-u)\} = \rho(u)$;
2. the conditional distribution of Y_i given $S(\cdot)$ is Gaussian with mean $\mu + S(x_i)$ and variance τ^2;
3. $Y_i : i = 1, \ldots, n$ are mutually independent, conditional on $S(\cdot)$.

Covariate information can be incorporated by assuming a non-constant mean $\mu = F\beta$ where F is a matrix with elements of the type $f_j(x_i)$, a measurement of the j^{th} covariate at the i^{th} location. The model can be made more flexible by incorporating the family of Box-Cox transformations (1), in which case the Gaussian model is assumed to hold for a transformation of the variable Y.

The basic model parameters are:

- β, the mean parameters,
- σ^2, the variance of the signal,
- τ^2, the variance of the noise,
- ϕ, the scale parameter of the correlation function.

Extra parameters provide greater flexibility:

- κ is an additional parameter, required by some models for correlation functions, which controls the smoothness of the field,
- (ψ_A, ψ_R) allows for geometric anisotropy,
- λ is the the Box-Cox transformation parameter.

Package features

The main features of the package are illustrated in the PDF document installed at 'geoR/docs/geoRintro.pdf'.

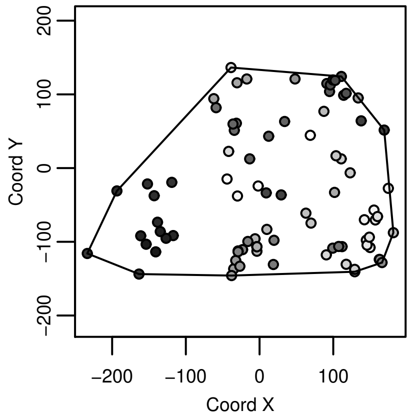


Figure 1: Output from the function `points.geodata` for the `wolfcamp` data.

There are functions available which can be used at different stages of geostatistical data analysis. Here we use the dataset `wolfcamp`, included in the package distribution, for a brief illustration of the package resources.

For **exploratory data analysis**, **geoR** uses R's graphical capabilities to produce plots of the data including their spatial locations. The most relevant functions are `plot.geodata` and `points.geodata`. Figure 1 shows an output from the latter.

Empirical variograms are used to explore the spatial structure of the data. Residual variograms can be obtained from an ordinary least squares "detrend", internal to the functions. Figure 2 shows directional and omni-directional variograms for the `wolfcamp` data assuming a first order polynomial trend on the coordinates. The main function for empirical variogram calculation is `variog`.

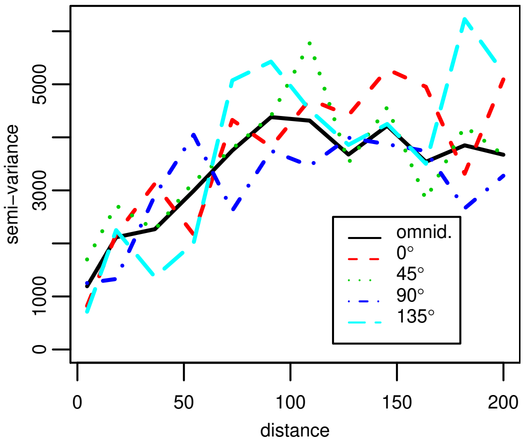


Figure 2: Directional and omni-directional variograms for the `wolfcamp` data.

Parameter estimation can be performed using different paradigms. Likelihood-based estimation (maximum and restricted maximum likelihood) is implemented by the function `likfit`. Alternatively, variogram-based estimation can be used. This consists of fitting a chosen parametric model to the sample variogram. The fitting can be done by "eye", ordinary least squares or weighted least squares. Figure 3 shows the empirical variogram and correlation function models fitted by different methods.

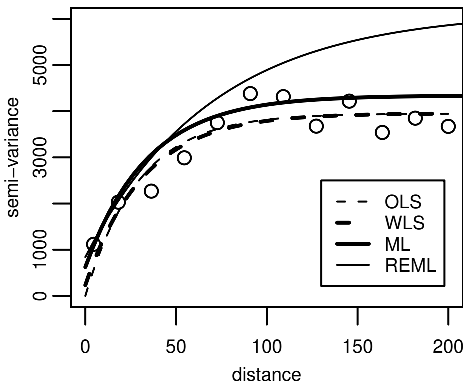


Figure 3: Directional and omni-directional variograms of the `wolfcamp` data.

The function `proflik` computes 1-D and 2-D profile likelihoods for the model parameters as illustrated by Figure 4. The profiles can be used to assess the variability of the estimates.

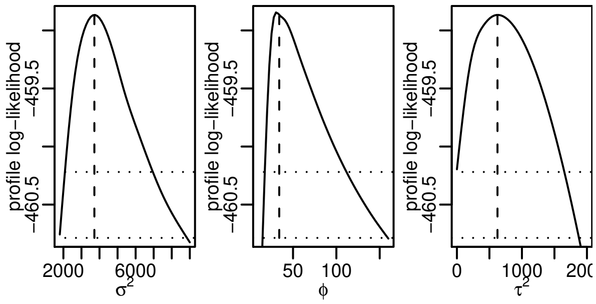


Figure 4: Profile likelihoods for the model parameters.

Spatial prediction by "plugging-in" the model parameters is performed by the function `krige.conv`. Depending on the input options the results correspond to methods known as *simple* , *ordinary, universal* and *external trend kriging*.

Model validation tools are implemented by the cross-validation function `xvalid`. Envelopes for the empirical variograms are computed by the functions `variog.model.env` and `variog.mc.env`.

Bayesian inference takes the parameter uncertainty into account when performing spatial prediction. Simulations of the posterior distribution $[S(x)|Y]$ allow inferences on linear and non-linear functionals of the signal $S(x)$ (Figure 5).

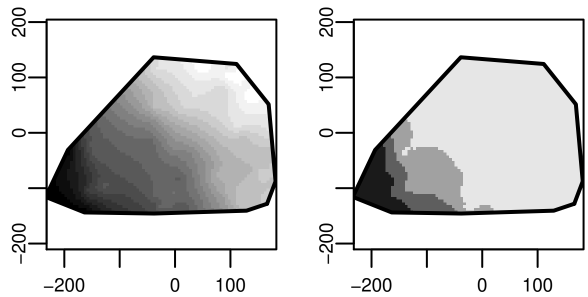


Figure 5: Predicted values over the area (left panel) and estimated probabilities of having values greater than 850 (right panel). The grey levels in right panel correspond to breaks $(0, 0.1, 0.5, 0.9, 1)$.

Simulation of (transformed) Gaussian random fields can be obtained by using the function `grf`. This function is intended for quick simulations with a small number of data locations. For simulations using large number of locations we recommend the package **RandomFields**.

Typically, the *methods* functionality is used to plot and/or print results returned by the main functions in the package. Additional details on the package resources can be found at the package's web-page and its documentation.

Future developments

The file 'docs/ToDo.geoR' is included in the package distribution and contains a list of features for which implementation is planned.

There is joint work with Ole Christensen (Aalborg University, Denmark) in progress to implement the non-Gaussian spatial models proposed by Diggle et al. (4). These models generalise the Gaussian model previously mentioned in a similar way that generalised linear models extend the classical linear regression model.

Acknowledgements

A special word of thanks goes to Ole Christensen who has reported the package usage in great detail and sent us a number of comments. We are grateful to several users who have reported usage, bugs and sent us suggestions. We also thank Martin Schlather and Patrick Brown for valuable discussions, and the *R Core Team*, in particular Brian Ripley and Kurt Hornik, for their support in making **geoR** available on CRAN.

Paulo acknowledges a grant from CAPES/Brazil. Peter acknowledges financial support from EPSRC and the TMR Network on Spatial and Computational Statistics.

Bibliography

[1] G. E. P. Box and D. R. Cox. An analysis of transformations (with discussion). *Journal of Royal Statistical Society series B*, 26:211–252, 1964. 16

[2] N. Cressie. *Statistics for Spatial Data – revised edition*. Wiley, New York, 1993.

[3] P. J. Diggle and P. J. Ribeiro Jr. Model-based geostatistics. XIV Sinape (Simpósio Nacional de Probabiulidade e Estatística, Caxambu, MG, Brasil, July 2000. 15

[4] P. J. Diggle, J. A. Tawn, and R. A. Moyeed. Model based geostatistics (with discussion). *Applied Statistics*, 47:299–350, 1998. 17

[5] D. G. Krige. A statistical approach to some mine valuations and allied problems at the Witwatersrand. Master's thesis, University of Witwatersrand, 1951. 15

[6] B. Matérn. *Spatial Variation*. Technical report, Statens Skogsforsningsinstitut, Stockholm, 1960. 15

[7] B. Matérn. *Spatial Variation*. Springer, Berlin, 2nd edition, 1986. 15

[8] G. Matheron. Principles of geostatistics. *Economic geology*, 58:1246–1266, 1963. 15

[9] B. D. Ripley. *Spatial Statistics*. Wiley, 1981. 15

[10] P. Whittle. On stationary processes in the plane. *Biometrika*, 41:434–449, 1954. 15

Paulo J. Ribeiro Jr
Universidade Federal do Paraná, Brasil
and *Lancaster University, UK*
Paulo.Ribeiro@est.ufpr.br

Peter J. Diggle
Lancaster University, UK
p.diggle@lancaster.ac.uk

Simulation and Analysis of Random Fields

by Martin Schlather

Random fields are the d-dimensional analogues of the one-dimensional stochastic processes; they are used to model spatial data as observed in environmental, atmospheric, and geological sciences. They are traditionally needed in mining and exploration to model ore deposits, oil reservoirs, and related structures.

The contributed package **RandomFields** allows for the simulation of Gaussian random fields defined on Euclidean spaces up to dimension 3. It includes some geostatistical tools and algorithms for the simulation of extreme-value random fields.

In the following two sections we give an example of an application, and a summary of the features of **RandomFields**.

A brief geostatistical analysis

To demonstrate key features we consider soil moisture data collected by the Soil Physics Group at the University of Bayreuth (see Figure 1), and perform a simple geostatistical analysis.

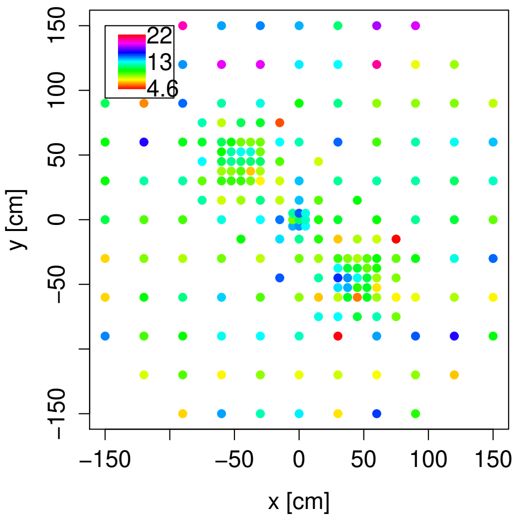


Figure 1: Measured soil moisture content (%)

The coordinates of the sample locations are assumed to be stored in `pts` and the moisture measurements in `d`. (See the example to the data set `soil` for the complete code.)

In geostatistics the variogram γ is frequently used to describe the spatial correlation structure. It can be expressed in terms of the (auto-)covariance function C if the random field is stationary: $\gamma(h) = C(0) - C(h)$ for $h \in \mathbb{R}^d$. We assume isotropy, i.e. γ depends only on the distance $|h|$. Then, we can find variogram values by

```
ev <- EmpiricalVariogram(pts, data=d,
        grid=FALSE,
        bin=c(-1, seq(0, 175, l=20)))
```

and select an appropriate model by

```
ShowModels(0:175, x=x, y=y, emp=ev)
```

see Figure 4 where `x` and `y` equal `seq(-150, 150, l=121)`. The parameters of the variogram model (here the Whittle-Matérn model) might be fitted by eye, but we prefer maximum likelihood,

```
p <- mleRF(pts, d, "whittle", param=rep(NA,5),
           lower.k=0.01, upper.k=30).
```

Now,

```
Kriging("O", x=x, y=y, grid=TRUE,
        model="whittle", par=p, given=pts,
        data=d)
```

yields the expected moisture content on the grid given by `x` and `y` (Figure 2).

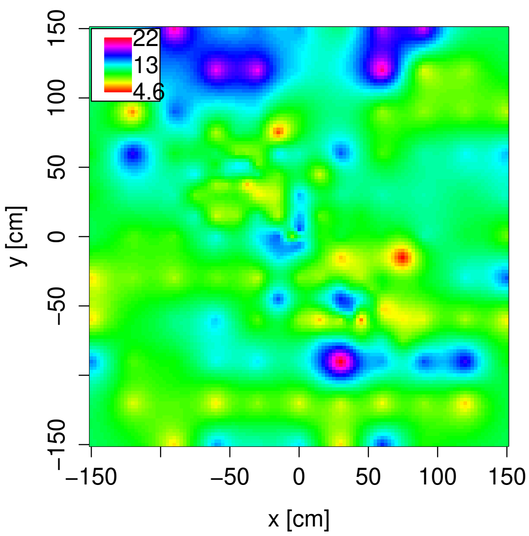


Figure 2: Kriged field

Conditional simulation, see Figure 3, allows for further inference.

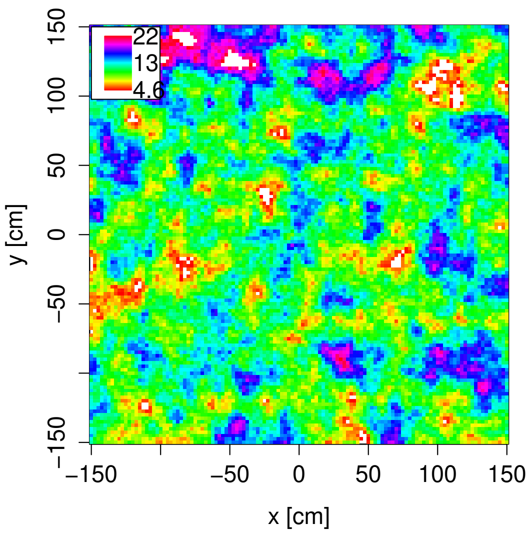


Figure 3: Conditional simulation

For instance, we may ask for the conditional probability, given the data at the respective locations, that the maximal moisture content is not greater than 24%. To this end, a random sample, here of size 100, is drawn from the conditional distribution

```
CondSimu("O", x=x, y=y, grid=TRUE, n=100,
         model="whittle", param=p, given=pts,
         data=d),
```

which yields an estimate of 40%.

Package features

The currently implemented methods for the simulation of stationary and isotropic random fields include circulant embedding, turning bands, and methods based on matrix decomposition or Poisson point processes. Important features of the package are

- User friendly interface: depending on his/her preferences, the user can either specify the desired simulation method, or the package will automatically pick an appropriate technique, according to the variogram model, the dimension of the space, and the spatial configuration of the simulation domain (1).

- Increased speed if multiple simulations with identical specifications are performed. To this end, a specification is first compared to that of the preceding simulation. In case the parameters match, the stored results of the deterministic part of the algorithm are used instead of being recalculated.

- The function `ShowModels` is an instructive tool for teaching, which allows for the interactive choice of variogram models and parameters. For each specification the graph of the variogram model and one- or two-dimensional simulations are provided. `ShowModels` can also be used to fit an empirical variogram by eye.

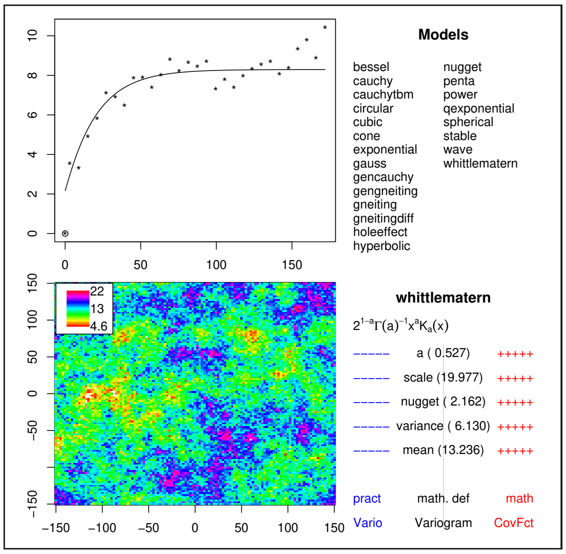


Figure 4: Snapshot of the interactive plot `ShowModels`

Further functionalities include:

- variogram analysis: calculation of the empirical variogram and MLE parameter estimation

- conditional simulation based on simple or ordinary kriging

- checks whether the parameter specifications are compatible with the variogram model

- simulation of max-stable random fields

Future extensions may provide further simulation algorithms for Gaussian and non-Gaussian random fields, and a basic toolbox for the analysis of geostatistical and spatial extreme value data.

Use `help(RandomFields)` to obtain the main man page. To start with, the examples in `help(GaussRF)` are recommended.

Acknowledgement. The work has been supported by the EU TMR network ERB-FMRX-CT96-0095 on "Computational and statistical methods for the analysis of spatial data" and the German Federal Ministry of Research and Technology (BMFT) grant PT BEO 51-0339476C. The author is thankful to Tilmann Gneiting, Martin Mächler, and Paulo Ribeiro for hints and discussions.

Bibliography

[1] M. Schlather. An introduction to positive definite functions and to unconditional simulation of random fields. Technical Report ST-99-10, Lancaster University, 1999. 19

Martin Schlather
University of Bayreuth, Germany
Martin.Schlather@uni-bayreuth.de

mgcv: GAMs and Generalized Ridge Regression for R

by Simon N. Wood

Generalized Additive Models (GAMs) have become quite popular as a result of the work of Wahba (1990) and co-workers and Hastie & Tibshirani (1990). Package **mgcv** provides tools for GAMs and other generalized ridge regression. This article describes how GAMs are implemented in **mgcv**: in particular the innovative features intended to improve the GAM approach. The package aims to provide the convenience of GAM modelling in S-PLUS, combined with much improved model selection methodology. Specifically, the degrees of freedom for each smooth term in the model are chosen simultaneously as part of model fitting by minimizing the Generalized Cross Validation (GCV) score of the whole model (not just component wise scores). At present **mgcv** only provides one dimensional smooths, but multi-dimensional smooths will be available from version 0.6, and future releases will include anisotropic smooths. GAMs as implemented in **mgcv** can be viewed as low rank approximations to (some of) the generalized spline models implemented in **gss** — the idea is to preserve most of the practical advantages with which elegant underlying theory endows the generalized smoothing spline approach, but without the formidable computational burden that accompanies full gss models of moderately large data sets.

GAMs in mgcv

GAMs are represented in **mgcv** as penalized generalized linear models (GLMs), where each smooth term of a GAM is represented using an appropriate set of basis functions and has an associated penalty measuring its wiggliness: the weight given to each penalty in the penalized likelihood is determined by its "smoothing parameter". Models are fitted by the usual iteratively re-weighted least squares scheme for GLMs, except that the least squares problem at each iterate is replaced by a penalized least squares problem, in which the set of smoothing parameters must be estimated alongside the other model parameters: the smoothing parameters are chosen by GCV. This section will sketch how this is done in a little more detail.

A GLM relating a univariate response variable y to a set of explanatory variables $x_1, x_2, \ldots$, has the general form:

$$g(\mu_i) = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \cdots \qquad (9.1)$$

where $E(y_i) \equiv \mu_i$ and the y_i are independent observations on r.v.s all from the same member of the exponential family. g is a smooth monotonic "link-function" that allows a useful degree of non-linearity into the model structure. The β_i are model parameters: likelihood theory provides the means for estimation and inference about them. The r.h.s. of (9.1) is the "linear predictor" of the GLM, and much of the statistician's modelling effort goes into finding an appropriate form for this.

The wide applicability of GLMs in part relates to the generality of the form of the of the linear predictor: the modeller is not restricted to including explanatory variables in their original form, but can include transformations of explanatory variables and dummy variables in whatever combinations are appropriate. Hence the class of models is very rich, including, for example, polynomial regression models and models for designed experiments. However the standard methods for generalized linear modelling can become unwieldy as models become more complex. In particular, it is sometimes the case that prior beliefs about appropriate model structure might best be summarized as something like:

$$g(\mu_i) = \beta_0 + s_1(x_{1i}) + s_2(x_{2i}) + \cdots \qquad (9.2)$$

i.e., the linear predictor should be given by a constant plus a smooth function of x_1 plus another smooth function of x_2 and so on (with some side conditions on the s_i to ensure identifiability). It is *possible* to build this sort of model structure directly within the GLM framework using, e.g. polynomials or more stable bases to represent the smooth terms: but such an approach becomes troublesome as the number of smooths and their complexity increases. The two main problems are that model selection becomes rather cumbersome (many models may need to be compared, and it is not always easy to keep them nested), and that the basis selected can have a rather strong influence on the fitted model (e.g. regression splines tend to be rather dependent on knot placement, while polynomials can be very unstable).

An alternative approach for working with models like (9.2) represents the smooth functions using linear smoothers, and performs estimation by backfitting (Hastie & Tibshirani, 1990) — this has the advantage that a very wide range of smoothers can be used, but the disadvantage that model selection (choosing the amount of smoothing to perform) is still difficult.

In **mgcv**, smooth terms in models like (9.2) are represented using penalized regression splines. That is, the smooth functions are re-written using a suitably chosen set of basis functions, and each has an associated penalty which enables its effective degrees of freedom to be controlled through a single smoothing parameter. How this works is best seen through an example, so consider a model with one linear term and a couple of smooth terms:

$$g(\mu_i) = \beta_0 + \beta_1 x_{1i} + s_1(x_{2i}) + s_2(x_{3i}) \qquad (9.3)$$

The s_i can be re-written in terms of basis functions thus:

$$s_1(x) = \sum_{j=1}^{k_1} \beta_{j+1} b_{1j}(x) \quad s_2(x) = \sum_{j=1}^{k_2} \beta_{j+1+k_1} b_{2j}(x)$$

where k_i is the number of basis functions used for s_i, the β_j are parameters to be estimated and the b_{ji} are basis functions. For example, a suitable set of spline-like basis functions might be:

$$b_{j1}(x) = x \text{ and } b_{ji}(x) = |x - x^*_{ji}|^3 \text{ for } i > 1$$

where the x^*_{ji} are a set of “knots” spread “nicely” throughout the relevant range of explanatory variable values. So (9.3) now becomes:

$$g(\mu_i) = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \beta_3 |x_{2i} - x^*_{22}|^3 + \cdots$$

… a GLM. If we write the vector of values of $g(\mu_i)$ as $\boldsymbol{\eta}$ then it's pretty clear that the previous equation written out for all i can be written as $\boldsymbol{\eta} = \mathbf{X}\boldsymbol{\beta}$, where the model matrix $\mathbf{X}$ follows in an obvious way from the above equation. (Note that **mgcv** actually uses a different (but equivalent) regression spline basis based on cubic Hermite polynomials: its parameters are usefully interpretable and it is computationally convenient, but rather long winded to write out.) So far the degrees of freedom associated with each smooth term are determined *entirely* by the k_i so that model selection will have all the difficulties alluded to above and the fitted model will tend to show characteristics dependent on knot locations. To avoid these difficulties **mgcv** uses a relatively high value for each k_i and controls the smoothness (and hence degrees of freedom) for each term through a set of penalties applied to the likelihood of the GLM. The penalties measure the wiggliness of each s_i as:

$$\int [s_i''(x)]^2 dx$$

Since $s_1''(x) = \sum_{j=1}^{k_1} \beta_{j+1} b''_{1j}(x)$, it's not hard to see that it is possible to write:

$$\int [s_1''(x)]^2 dx = \boldsymbol{\beta}^T \mathbf{S}_1 \boldsymbol{\beta}$$

where $\boldsymbol{\beta}$ is the parameter vector, and $\mathbf{S}_1$ is a positive semi-definite matrix depending only on the basis functions. A similar result applies to s_2. So the model β_i's can be estimated by minimizing:

$$-l(\boldsymbol{\beta}) + \sum_{i=1}^{2} \lambda_i \boldsymbol{\beta}^T \mathbf{S}_i \boldsymbol{\beta}$$

where l is the log-likelihood for $\boldsymbol{\beta}$, and the λ_i's control the relative weight given to the conflicting goals of good fit and model smoothness. *Given* λ_i it is straightforward to solve this problem by (penalized) IRLS, but the λ_i need to be estimated, and this is not so straightforward.

Recall that the IRLS method for a GLM consists of iterating the following steps to convergence:

1. The current estimate of $\boldsymbol{\beta}$, $\boldsymbol{\beta}^{[k]}$, yields estimates of $\boldsymbol{\mu}$ and the variance of each y_i: V_i. Hence using the current estimates, calculate the following: (i) the diagonal weight matrix $\mathbf{W}$ where:

   $$W_{ii} = [g'(\mu_i)^2 V_i]^{-1}$$

   and (ii) the vector of pseudodata:

   $$\mathbf{z} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\Gamma}(\mathbf{y} - \boldsymbol{\mu})$$

   where $\boldsymbol{\Gamma}$ is a diagonal matrix and $\Gamma_{ii} = [g'(\mu_i)]^{-1}$.

2. Minimize:

$$\|\mathbf{W}^{1/2}(\mathbf{z}-\mathbf{X}\boldsymbol{\beta})\|^2$$

w.r.t. $\boldsymbol{\beta}$ to get $\boldsymbol{\beta}^{[k+1]}$.

mgcv fits GAMs by replacing step 2. with the following:

2. Find the λ_i minimizing:

$$\frac{\|\mathbf{W}^{1/2}(\mathbf{z}-\mathbf{X}\boldsymbol{\beta})\|^2}{[\mathrm{tr}(\mathbf{I}-\mathbf{A})]^2} \tag{9.4}$$

where $\boldsymbol{\beta}$ is the solution to the problem of minimizing:

$$\|\mathbf{W}^{1/2}(\mathbf{z}-\mathbf{X}\boldsymbol{\beta})\|^2 + \sum \lambda_j \boldsymbol{\beta}^T \mathbf{S}_j \boldsymbol{\beta}$$

w.r.t. $\boldsymbol{\beta}$, and $\mathbf{A}$ is the "influence" or "hat" matrix: $\mathbf{X}(\mathbf{X}^T\mathbf{W}\mathbf{X} + \sum \lambda_j \boldsymbol{\beta}^T \mathbf{S}_j \boldsymbol{\beta})^{-1}\mathbf{X}^T\mathbf{W}$, whose trace gives the estimated degrees of freedom for the model.

(9.4) is the GCV score for the model and its efficient minimization is the key to the approach used in **mgcv**: the method for doing this is based on a generalization of the method developed by Gu & Wahba (1991) for generalized smoothing spline models, and is described in Wood (2000). The computational burden is cubic in the dimension of $\boldsymbol{\beta}$ — which is usually much less than the computational burden of using Gu and Wahba's method for gss models, which is necessarily cubic in the number of data.

Note that direct minimization of (9.4) is not the same as minimizing separate GCV scores as part of each back-fitting iteration — the latter approach is very difficult to justify on other than ad hoc grounds.

A simple Bayesian argument yields a covariance matrix estimate for $\hat{\boldsymbol{\beta}}$ and hence estimated Bayesian confidence intervals for the components of the GAM (Wood, 2000; Hastie & Tibshirani, 1990). These are similar in spirit to the intervals for smoothing splines in Wahba (1983).

Note then, that the key point about **mgcv** is that the selection of degrees of freedom for the components of a fitted GAM is an integral part of model fitting. Furthermore the manner in which it is integrated is designed to make inclusion of multidimensional and anisotropic smooths quite straightforward. In addition it should be clear that in principle any smooth constructed using a basis and quadratic penalty could be incorporated into **mgcv**'s GAM modelling tools.

Practical GAMs

Because of the way in which estimation of degrees of freedom is integrated into model fitting, the `gam()` function provided by **mgcv** is not an exact clone of what is described in the white book and implemented in S-PLUS. This section describes what is implemented and how to use it.

`gam()`

mgcv's `gam()` function fits a GAM specified by a model formula and family, to univariate response data. A simple example of its use is:

```
> gam(y ~ s(x))
```

which will cause the model:

$$y_i \sim f(x_i) + \epsilon_i, \quad \epsilon_i \text{ i.i.d. } N(0, \sigma^2)$$

to be estimated, where f is a smooth function.

A more complicated example, illustrating a few more features is:

```
> gam(y^0.5 ~ -1 + x + s(z,5|f) + s(w) + s(v,20),
    data = mydata, family = gamma(link=I))
```

In this case the response is $\sqrt{y}$, and the linear predictor is made up of a linear term in x plus smooth functions of z, w and v, with no intercept term. The data are assumed to follow a gamma distribution, and the link is the identity function. The 3 different forms of `s()` each relate to a different representation of the smooth concerned. `s(z,5|f)` indicates that the smooth function of z is to be represented as a pure un-penalized regression spline, with 5 knots — under the representation used by **mgcv** this corresponds to exactly 4 degrees of freedom ('`|f`' indicates *f*ixed degrees of freedom). `s(w)` indicates that the smooth function of w is to be represented using a default 10 knot penalized regression spline: corresponding to maximum degrees of freedom of 9 for this term. Finally `s(v,20)` indicates that the smooth of v is to be represented by a 20 knot penalized regression spline: this term is being allowed a maximum of 19 degrees of freedom — presumably the relationship between $\sqrt{y}$ and v is expected to be fairly complicated.

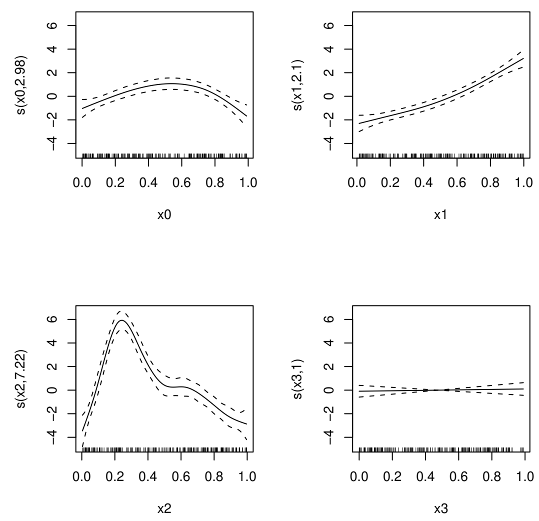
The choice of the number of knots is not very critical, but should be somewhat larger than the estimated degrees of freedom plus 1, otherwise: (i) the choice of knot locations will begin to have a visible effect on the shape of the estimated function, and (ii) it is *possible* that if the GCV function is a strange shape then the optimization path to the GCV minimum may pass beyond the upper boundary on degrees of freedom, so that the smoothing parameter estimates become incorrectly stuck at that boundary. In practice I usually start with the default 10 knot splines, but increase the number of knots for any terms that are then estimated to have close to the corresponding maximum 9 degrees of freedom.

Other arguments to `gam()` will be familiar to users of `lm()` and `glm()`. The exception is the argument `scale`. If the scale parameter for the model distribution is known then there is an argument for replacing GCV with the unbiased risk estimator (UBRE) given e.g. in Wahba (1990). Hence if `scale` is a positive number it is assumed to be the scale parameter and UBRE is used. If `scale` is zero (the default) then UBRE is used for the Poisson and binomial distributions (scale parameter 1), but GCV is used otherwise. A negative value for the scale parameter forces use of GCV in all cases. Note that GCV is appropriate if over-dispersion is suspected.

Other GAM functions

Package **mgcv** provides versions of `print.gam()`, `predict.gam()` and `plot.gam()`. `plot.gam()` differs most from what is available in S-PLUS — interactive use is missing, for example. Here is an example of its use to plot the results of fitting a simple 4 term model:

```
> gam.model <- gam(y ~ s(x0)+s(x1)+s(x2)+s(x3))
> plot(gam.model, pages = 1)
```



By default the same y axis range is used for all the plots, while the `pages=1` option provides automatic layout on a single page. The rug plots at the foot of each plot show the observed values of each explanatory variable. Each y-axis label indicates what the smooth is a function of and how many degrees of freedom the term has. The solid line in each plot is the estimate of the smooth function, while the dashed lines are at 2 standard errors above and below the estimate — roughly 95% confidence limits. This example used simulated data and `x3` is in fact unrelated to the response: notice how the smooth for `x3` is estimated to have just one degree of freedom, and how its "confidence band" comfortably includes zero everywhere.

It's also of some interest to print the `gam.model`:

```
> gam.model

Family: gaussian
Link function: identity

Formula:
y ~ s(x0) + s(x1) + s(x2) + s(x3)

Estimated degrees of freedom:
 2.982494 2.096610 7.219753 1.000005
 total =  14.29886

GCV score:  4.326104
```

The estimated degrees of freedom for each smooth term are given in the order in which the smooth terms are specified in the model formula — note that the total degrees of freedom includes those associated with purely parametric model components. The GCV score is useful for comparing models with and without particular terms included.

Dropping model terms

While **mgcv** selects the degrees of freedom for each term automatically, the nature of the estimation algorithm means that it can not automatically decide whether to drop a term altogether or not. The reason for this is that a zero term and a straight line have the same zero penalty — hence once a term has become a straight line, increasing its smoothing parameter further can have no effect on its degrees of freedom, and certainly won't force it to become zero. Hence the modeller must remove unwanted terms "by hand". Deciding which terms to remove is straightforward, and should be guided by the answers to 3 questions:

1. Is the estimated degrees of freedom for the term close to 1?

2. Does the plotted confidence band for the term include zero everywhere?

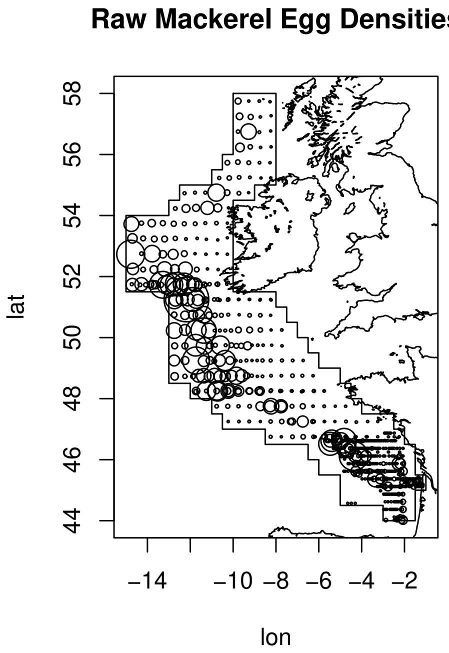
3. Does the GCV score drop when the term is dropped?

If the answer to all three questions is "yes" then the term should be dropped. If the e.d.f. is close to one but the answer to the other 2 questions is "no" then you might as well replace the smooth with a parametric linear term. Other cases will require judgement: for example, very small increases in GCV score shouldn't prevent a term from being dropped. Because of correlations between explanatory variable, terms should only be dropped one at a time: it makes sense to start with the term for which the zero line is most comfortably within the confidence band.

In the plot shown above it's clear that `x3` should be dropped from the model: the example in the next section provides a second illustration.

A fisheries example

As an example of use of GAMs with **mgcv**, consider a set of data originally analysed by Borchers *et al.* (1997) as part of the stock assessment process for the European mackerel fishery.



The data are mackerel egg densities (per m^2 of sea surface) obtained from net hauls undertaken from research boats in 1992 (Bowman & Azzalini, 1997, analyse a subset of these data, with slightly different pre-processing, using a loess model — their subset is available in package **sm**). The plotted symbol sizes are proportional to the egg density. Candidate explanatory variables in this case are longitude, latitude, sea bed depth, sea surface temperature and distance from the 200 metre sea bed depth contour. If a variance stabilizing transformation is employed then a Gaussian error model is not too bad. So, a first model attempt might be:

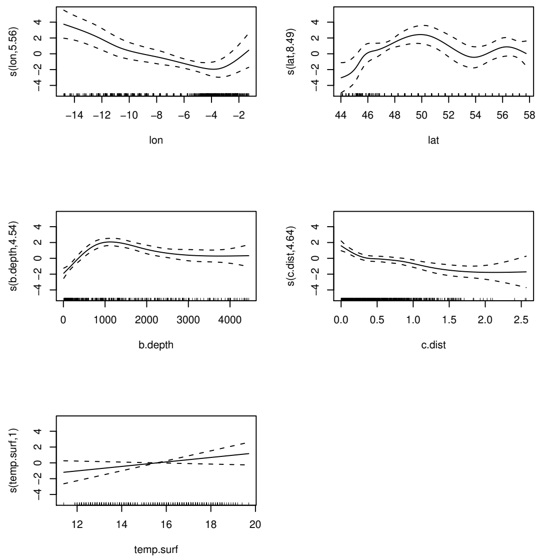
```
> mack.fit <-
+ gam(egg.dens^0.4 ~ s(lon) + s(lat) + s(b.depth)
+       + s(c.dist) + s(temp.surf),
+     data = mack)
```

(data frame `mack` contains the data for the 634 sampling stations). Here are the first results (the fitting took a few seconds on a Pentium II):

```
> mack.fit

Family: gaussian
Link function: identity

Formula:
egg.dens^0.4 ~ s(lon) + s(lat) +
s(b.depth) + s(c.dist) + s(temp.surf)

Estimated degrees of freedom:
 5.55544 8.494712 4.536643 4.63995
 1.000001   total =  25.22675

GCV score:  3.71533
```

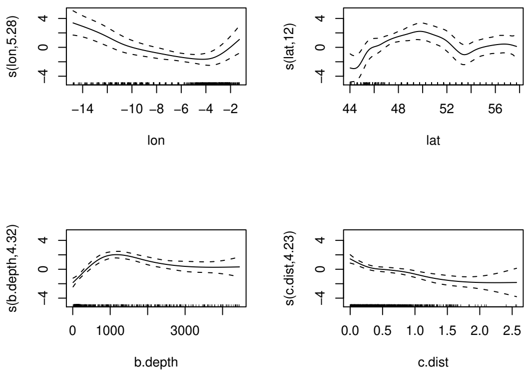
clearly surface temperature is a candidate for removal or replacement by a linear term: the next thing to do is to plot the estimated terms:

```
> plot(mack.fit)
```



So, surface temperature looks like a candidate for removal (and at the same time it's wise to increase the number knots for the latitude term).

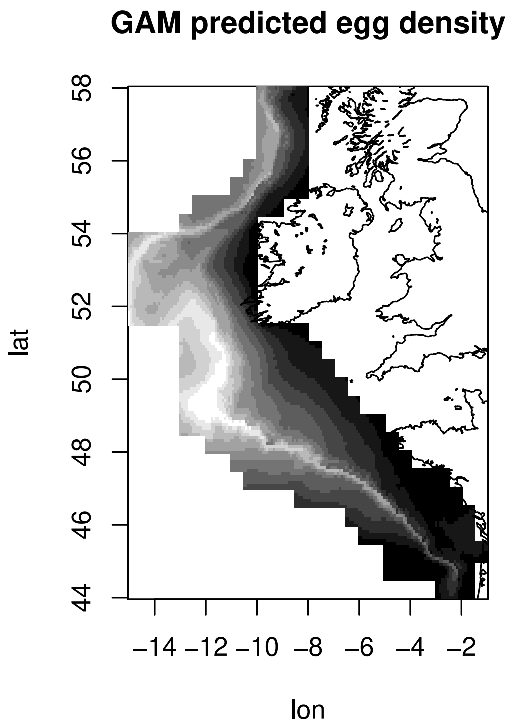
```
> mack.fit2 <-
+ gam(egg.dens^0.4 ~ s(lon) + s(lat, 20)
+       + s(b.depth) + s(c.dist),
+     data = mack)
> mack.fit2

Family: gaussian
Link function: identity

Formula:
egg.dens^0.4 ~ s(lon) + s(lat, 20) +
s(b.depth) + s(c.dist)

Estimated degrees of freedom:
 5.276965 12.00392 4.323457 4.234603
 total =  26.83895

GCV score:  3.709722

> plot(mack.fit2, pages = 1)
```



The GCV score has decreased, supporting the decision to remove the surface temperature term. There being no further terms to delete, the model can be used for prediction. Data frame `mackp` contained the explanatory variables on a fine grid over the survey area. To get a predicted (transformed) density for each point on this grid I used `predict.gam()`:

```
> mack.pred <- predict.gam(mack.fit2, mackp)
```

A certain amount of tedious manipulation was needed to copy this into a matrix suitable for contouring and plotting, but an image plot of the final predicted (transformed) densities looks like this:



Summary and the future

mgcv offers GAM modelling tools with much (but not all) of the functionality of their S-PLUS equivalents, plus the substantial advantage of well founded and efficient methods for selecting the degrees of freedom for each smooth term (and for selecting which terms to include at all). The package currently offers a more limited range of models than the **gss** package, but typically at much lower computational cost, and with slightly greater ease of use for those familiar with GAMs in S-PLUS. Future developments will provide further basic methods, and two further substantial enhancements. Version 0.6 will include multi-dimensional smooth terms, while in the longer term anisotropic smooth terms will be included.

References

Borchers, D. L., S. T. Buckland, I. G. Priede and S. Ahmadi (1997). Improving the precision of the daily egg production method using generalized additive models. *Canadian Journal of Fisheries and Aquatic Science*, **54**, 2727–2742.

Bowman, A. W. and A. Azzalini (1997). *Applied Smoothing Techniques for Data Analysis: The Kernel Approach with S-Plus Illustrations*. Clarendon Press.

Gu C. and G. Wahba (1991) Minimizing GCV/GML scores with multiple smoothing parameters via the newton method. *SIAM Journal on Scientific and Statistical Computation*, **12**, 383–398.

Hastie T. J. and R. J. Tibshirani (1990). *Generalized Additive Models*. Chapman & Hall.

Wahba, G. (1983). Bayesian confidence intervals for the cross validated smoothing spline. *Journal of the Royal Statistical Society (B)*, **45**, 133–150.

Wahba, G. (1990). *Spline Models for Observational Data*. SIAM, Philadelphia.

Wood, S. N. (2000). Modelling and smoothing parameter estimation with multiple quadratic penalties. *Journal of the Royal Statistical Society (B)*, **62**, 413–428.

Simon N. Wood
University of St Andrews, Fife, UK
snw@mcs.st-and.ac.uk

What's the Point of 'tensor'?

by Jonathan Rougier

One of the packages on CRAN is called **tensor**, and it exists to compute tensor products of arrays. The tensor product is a generalisation of the matrix product, but you can go a lot further if you want: to experience the full power of tensor methods in statistics you should consult McCullagh (1987), not for the faint-hearted.

Tensor notation

In statistics it is often necessary to compute measures based on linear combinations of vectors of uncertain quantities. For example, if x and y are uncertain we might like to know $\mathsf{Cov}[Ax, By]$ where A and B are known matrices. In this case we know that the answer in matrix-form is $A\,\mathsf{Cov}[x, y]\,B^{\mathsf{T}}$.

Tensors provide an alternative to the matrix-form for representing linear combinations. The convention with tensors is to use a representitive component to stand for the whole object, and to let repeated indices denote sums of products across objects. Thus instead of writing $v = Ax$ in matrix-form, we would write $v_i = A_{ij}\,x_j$ in tensor-form, by which we mean $v_i = \sum_j a_{ij}\,x_j$, for $i = 1, \ldots, m$ where A has m rows.

The great advantage of tensors is their transparency in use with linear operators. To compute $\mathsf{Cov}[Ax, By]$, for example, first write it as a tensor then treat everything as scalars:

$$\begin{aligned}\mathsf{Cov}[Ax, By] &= \mathsf{Cov}[A_{ij}\,x_j, B_{k\ell}\,y_\ell] \\ &= A_{ij}\,\mathsf{Cov}[x_j, y_\ell]\,B_{k\ell}.\end{aligned}$$

To do a tensor calculation the `tensor` function has to know what the objects are and what their common indices are. Using `t` and `%*%` would be faster here, but in tensor-form we might do `tensor(tensor(A, Cxy, 2, 1), B, 2, 2)`. The inner call zips up the j index between A_{ij} and $\mathsf{Cov}[x_j, y_\ell]$, and creates an object with index set $i\ell$, and the outer call zips up the ℓ index with $B_{k\ell}$ to leave an object with index set ik. Another way to do the calculation would be `tensor(A %o% B, Cxy, c(2, 4), c(1, 2))`, where '`%o%`' computes the outer product (this is still two tensor products but only one is apparent: the outer product is a degenerate tensor product with no summing). This way creates an object with index set $ijk\ell$ and then zips up $j\ell$ with $\mathsf{Cov}[x_j, y_\ell]$.

Here's another example of the transparency of tensor-form. I am always forgetting the expectation of the quadratic form $x^{\mathsf{T}}Ax$ in terms of the mean μ and variance Σ of x, which, for reference, is usually written $\mathsf{E}[x^{\mathsf{T}}Ax] = \mu^{\mathsf{T}}A\mu + \mathrm{tr}(A\Sigma)$. Write the quadratic form as a tensor, however, and we see that

$$\begin{aligned}\mathsf{E}[x^{\mathsf{T}}Ax] &= \mathsf{E}[x_i\,A_{ii'}\,x_{i'}] \\ &= A_{ii'}\,\mathsf{E}[x_i\,x_{i'}] \\ &= A_{ii'}\,\{\mu_i\,\mu_{i'} + \Sigma_{ii'}\}\,.\end{aligned}$$

So although you might be tempted by `t(m) %*% A %*% m + sum(diag(A %*% S))` from the textbook, the smart money is on `sum(A * (m %o% m + S))`. We do not need the `tensor` function here, but it would be `tensor(A, m %o% m + S, c(1, 2), c(1, 2))`.

Rectangular objects

The greatest strength of tensor-form occurs in the generalisation from matrices to arrays. Often uncertain objects possess a rectangular structure, e.g. $X = \{x_{ijk}\}$ might have a natural three-dimensional structure. The mean of X is an array of the same shape as X, while the variance of X is an array with the same shape as the outer product of X with itself. Thus the variance of X_{ijk} is, in tensor form, the six-dimensional object

$$\begin{aligned}C_{ijki'j'k'} &= \mathsf{Cov}[X_{ijk}, X_{i'j'k'}] \\ &= \mathsf{E}[x_{ijk}\,x_{i'j'k'}] - \mathsf{E}[x_{ijk}]\,\mathsf{E}[x_{i'j'k'}]\end{aligned}$$

where primed indices are used to indicate similarity.

Even if X is two-dimensional (i.e. a matrix), sticking with the matrix-form can cause problems. For example, to compute $\mathsf{Cov}[AXB, X]$ we might use the identity $\mathrm{vec}(AXB) = (B^{\mathsf{T}} \otimes A)\,\mathrm{vec}\,X$, where '$\otimes$' denotes the kronecker product and '$\mathrm{vec}\,X$' creates a vector by stacking the columns of X. Originally, that is why I contributed the function `kronecker` to R. Then we would have

$$\mathsf{Cov}[\,\mathrm{vec}(AXB), \mathrm{vec}\,X] = (B^{\mathsf{T}} \otimes A)\,\mathsf{Var}\,[\,\mathrm{vec}\,X]$$

which requires us to vectorise the naturally two-dimensional object X to give us a two-dimensional representation of the naturally four-dimensional object $\mathsf{Var}\,[X]$. In tensor-form we would have instead

$$\mathsf{Cov}[A_{ij}\,X_{jk}\,B_{kl}, X_{j'k'}] = A_{ij}\,B_{kl}\,\mathsf{Cov}[X_{jk}, X_{j'k'}]$$

where $\mathsf{Cov}[X_{jk}, X_{j'k'}]$ preserves the natural shape of the variance of X. We might compute this by

```
tmp <- tensor(tensor(A, varX, 2, 1), B, 2, 1)
aperm(tmp, c(1, 4, 2, 3))
```

The temporary object `tmp` has index set $ij'k'l$ which is permuted using `aperm` to the set $ilj'k'$. The same result could be achieved more spectacularly using

```
tensor(A %o% B, varX, c(2, 3), c(1, 2))
```

It is often the case that you can arrange for the index set to come out in the right order, and so remove the need for a call to `aperm`.

When the uncertain objects are more than two-dimensional then tensors are usually the only option, both for writing down the required calculation, and performing it. You can find an example in the Appendix of our forthcoming paper (Craig *et al*, 2001) which is expressed entirely in tensor-form. In this paper we have a mixture of one-, two- and three-dimensional objects which are also very large. For example, lurking in the Appendix you can find expressions such as

$$\mathsf{Cov}\left[B_{im},\, S_{i''r}\right]\, P_{i''ri'''r'}\, H^{\epsilon}_{i'i'''r'}(x)\, g_m(x),$$

a two-dimensional expression (in ii') which is a function of the vector x. To give you some idea of the scale of this calculation, there are about 60 i's, 45 m's, and 6 r's, and x is 40-dimensional. (We then have to integrate x out of this, but that's another story!).

In our current development of this work we will need to evaluate the object $\mathsf{E}\left[B_{im}\, B_{i'm'}\, B_{i''m''}\, B_{i'''m'''}\right]$. This has about 5×10^{13} components and so this calculation is currently 'on hold', but I am hoping that Luke Tierney's eagerly anticipated *hyper-spatial* memory manager will sort this out and spare me from having to thinking too deeply about the essential structure of these objects.

Implementation

Conceptually, a tensor product is a simple thing. Take two arrays `A` and `B`, and identify the extents in each to be collapsed together (make sure they match!). Permute the collapse extents of `A` to the end and the collapse extents of `B` to the beginning. Then re-shape both `A` and `B` as matrices and take the matrix product. Reshape the product to have the non-collapse extents of `A` followed by the non-collapse extents of `B` and you are done.

In order for these operations to be efficient, we need a rapid `aperm` operation. The `aperm` function in R 1.2 was not really fast enough for large objects, and so I originally wrote **tensor** entirely in `C`, using a Python approach for arrays called 'strides'. In R 1.3 however, there is a new `aperm` that itself uses strides, to achieve a reduction in calculation time of about 80%. The new version of **tensor**, version 1.2, therefore does the obvious thing as outlined above, and does it entirely in R. But the old version of **tensor**, version 1.1-2, might still be useful for dealing with very large objects, as it avoids the duplication that can happen when passing function arguments.

References

P. S. Craig, M. Goldstein, J. C. Rougier and A. H. Seheult (2001). Bayesian forecasting for complex systems using computer simulators, *Journal of the American Statistical Association*, forthcoming.

P. McCullagh (1987). *Tensor Methods in Statistics*, Chapman and Hall.

Jonathan Rougier
University of Durham, UK
J.C.Rougier@durham.ac.uk

On Multivariate *t* and Gauß Probabilities in R

by Torsten Hothorn, Frank Bretz and Alan Genz

Introduction

The numerical computation of a multivariate normal or t probability is often a difficult problem. Recent developments resulted in algorithms for the fast computation of those probabilities for arbitrary correlation structures. We refer to the work described in (3), (4) and (2). The procedures proposed in those papers are implemented in package **mvtnorm**, available at CRAN. Basically, the package implements two functions: `pmvnorm` for the computation of multivariate normal probabilities and `pmvt` for the computation of multivariate t probabilities, in both cases for arbitrary means (resp. noncentrality parameters), correlation matrices and hyperrectangular integration regions.

We first illustrate the use of the package using a simple example of the multivariate normal distribution. A little more details are given in Section 'Details'. Finally, an application of `pmvt` in a multiple testing problem is discussed.

A simple example

Assume that $X = (X_1, X_2, X_3)$ is multivariate normal with correlation matrix

$$\mathbf{\Sigma} = \begin{pmatrix} 1 & \frac{3}{5} & \frac{1}{3} \\ \frac{3}{5} & 1 & \frac{11}{15} \\ \frac{1}{3} & \frac{11}{15} & 1 \end{pmatrix}$$

and expectation $\mu = (0, 0, 0)^\top$. We are interested in the probability

$$P(-\infty < X_1 \leq 1, -\infty < X_2 \leq 4, -\infty < X_3 \leq 2).$$

This is computed as follows:

```
R> m <- 3
R> sigma <- diag(3)
R> sigma[2,1] <- 3/5
R> sigma[3,1] <- 1/3
R> sigma[3,2] <- 11/15
R> pmvnorm(lower=rep(-Inf, m), upper=c(1,4,2),
           mean=rep(0, m), sigma)
$value
[1] 0.8279847

$error
[1] 5.684033e-07

$msg
[1] "Normal Completion"
```

The mean vector is passed to `pmvnorm` by the argument `mean`, and `sigma` is the correlation matrix (only the lower triangle being used) The region of integration is given by the vectors `lower` and `upper`, both can have elements `-Inf` or `+Inf`. The value of `pmvnorm` is a list with the following components:

`value`: the estimated integral value,

`error`: the estimated absolute error,

`msg`: a status message, indicating whether or not the algorithm terminated correctly.

From the results above it follows that $P(-\infty < X_1 \leq 1, -\infty < X_2 \leq 4, -\infty < X_3 \leq 2) \approx 0.82798$ with an absolute error estimate of 2.7×10^{-7}.

Details

This section outlines the basic ideas of the algorithms used. The multivariate t distribution (MVT) is given by

$$\mathbf{T}(\mathbf{a}, \mathbf{b}, \boldsymbol{\Sigma}, \nu) = \frac{2^{1-\frac{\nu}{2}}}{\Gamma(\frac{\nu}{2})} \int_0^\infty s^{\nu-1} e^{-\frac{s^2}{2}} \boldsymbol{\Phi}\left(\frac{s\mathbf{a}}{\sqrt{\nu}}, \frac{s\mathbf{b}}{\sqrt{\nu}}, \boldsymbol{\Sigma}\right) ds,$$

where the multivariate normal distribution function (MVN)

$$\boldsymbol{\Phi}(\mathbf{a}, \mathbf{b}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{|\boldsymbol{\Sigma}|(2\pi)^m}} \int_{a_1}^{b_1} \int_{a_2}^{b_2} \ldots \int_{a_m}^{b_m} e^{-\frac{1}{2}\mathbf{x}^t \boldsymbol{\Sigma}^{-1}\mathbf{x}} d\mathbf{x},$$

$\mathbf{x} = (x_1, x_2, ..., x_m)^t$, $-\infty \leq a_i < b_i \leq \infty$ for all i, and $\boldsymbol{\Sigma}$ is a positive semi-definite symmetric $m \times m$ matrix. The original integral over an arbitrary m-dimensional, possibly unbounded hyper-rectangle is transformed to an integral over the unit hypercube.

These transformations are described in (3) for the MVN case and in (2) for the MVT case. Several suitable standard integration routines can be applied to this transformed integral. For the present implementation randomized lattice rules were used. Such lattice rules seek to fill the integration region evenly in a deterministic process. In principle, they construct regular patterns, such that the projections of the integration points onto each axis produce an equidistant subdivision of the axis. Robust integration error bounds are obtained by introducing additional shifts of the entire set of integration nodes in random directions. Since this additional randomization step is only performed to introduce a robust Monte Carlo error bound, 10 simulation runs are usually sufficient. For a more detailed description (2) might be referred to.

Applications

The multivariate t distribution is applicable in a wide field of multiple testing problems. We will illustrate this using a example studied earlier by (1). For short, the effects of 5 different perfusates in capillary permeability in cats was investigated by (5). The data met the assumptions of a standard one-factor ANOVA. For experimental reasons, the investigators were interested in a simultaneous confidence intervals for the following pairwise comparisons: $\beta_1 - \beta_2, \beta_1 - \beta_3, \beta_1 - \beta_5, \beta_4 - \beta_2$ and $\beta_4 - \beta_3$. Therefore, the matrix of contrast is given by

$$\mathbf{C} = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 1 & -1 & 0 \end{pmatrix}.$$

Reference (1) assumed that $\beta = (\beta_1, \ldots, \beta_5)$ is multivariate normal with mean β and covariance matrix $\sigma^2\mathbf{V}$, where $\mathbf{V}$ is known. Under the null hypothesis $\beta = 0$, we need knowledge about the distribution of the statistic

$$W = \max_{1 \leq j \leq 5} \left\{ \frac{|c_j(\hat{\beta} - \beta)|}{\hat{\sigma}\sqrt{c_j \mathbf{V} c_j^\top}} \right\}$$

where c_j is the jth row of $\mathbf{C}$. By assumption, $\hat{\sigma}$ is χ_ν distributed, so under hypothesis W the argument to max follows a multivariate t distribution. Confidence intervals can be obtained by $c_j\hat{\beta} \pm w_\alpha \hat{\sigma}\sqrt{c_j \mathbf{V} c_i^\top}$, where w_α is the $1 - \alpha$ quantile of the null distribution of W. Using `pmvt`, one can easily compute the quantile for the example cited above.

```
R> n <- c(26, 24, 20, 33, 32)
R> V <- diag(1/n)
R> df <- 130
R> C <- c(1,1,1,0,0,-1,0,0,1,0,0,-1,0,0,
```

```
        1,0,0,0,-1,-1,0,0,-1,0,0)
R> C <- matrix(C, ncol=5)
R> cv <- C %*% V %*% t(C)
R> cr <- matrix(rep(0, ncol(cv)^2),
              ncol=ncol(cv))
R> for (i in 1:5) {
     for (j in 1:5) {
       cr[i,j] <- cv[i,j] / sqrt(cv[i,i]*cv[j,j])
     }
   }
R> delta <- rep(0,5)
R> myfct <- function(q, alpha) {
     lower <- rep(-q, ncol(cv))
     upper <- rep(q, ncol(cv))
     pmvt(lower, upper, df, cr, delta,
          abseps=0.0001)$value - alpha
   }
R> round(uniroot(myfct, lower=1, upper=5,
               alpha=0.95)$root, 3)
[1] 2.561
```

Here `n` is the sample size vector of each level of the factor, `V` is the covariance matrix of β. With the contrasts $\mathbf{C}$ we can compute the correlation matrix `cr` of $\mathbf{C}\beta$. Finally, we are interested in the 95% quantile of W. A wrapper function `myfct` computes the difference of the multivariate t probability for quantile `q` and `alpha`. The `alpha` quantile can now be computed easily using `uniroot`. The 95% quantile of W in this example is 2.561; reference (1) obtained 2.562 using 80.000 simulation runs. The computation needs 8.06 seconds total time on a Pentium III 450 MHz with 256 MB memory.

Using package **mvtnorm**, the efficient computation of multivariate normal or t probabilities is now available in R. We hope that this is helpful to users / programmers who deal with multiple testing problems.

Bibliography

[1] Don Edwards and Jack J. Berry. The efficiency of simulation-based multiple comparisons. *Biometrics*, 43:913–928, December 1987. 28, 29

[2] A. Genz and F. Bretz. Numerical computation of multivariate t-probabilities with application to power calculation of multiple contrasts. *Journal of Statistical Computation and Simulation*, 63:361–378, 1999. 27, 28

[3] Alan Genz. Numerical computation of multivariate normal probabilities. *Journal of Computational and Graphical Statistics*, 1:141–149, 1992. 27, 28

[4] Alan Genz. Comparison of methods for the computation of multivariate normal probabilities. *Computing Science and Statistics*, 25:400–405, 1993. 27

[5] P. D. Watson, M. B. Wolf, and I. S. Beck-Montgemery. Blood and isoproterenol reduce capillary permeability in cat hindlimb. *The American Journal of Physiology*, 252:H47–H53, 1987. 28

Torsten Hothorn
Friedrich-Alexander-Universität Erlangen-Nürnberg
Institut für Medizininformatik, Biometrie und Epidemiologie
Waldstraße 6, D-91054 Erlangen
Torsten.Hothorn@rzmail.uni-erlangen.de

Frank Bretz
Universität Hannover
LG Bioinformatik, FB Gartenbau
Herrenhäuser Str. 2
D-30419 Hannover
bretz@ifgb.uni-hannover.de

Alan Genz
Department of Mathematics
Washington State University
Pullman, WA 99164-3113 USA
alangenz@wsu.edu

The first author gratefully acknowledges support by Deutsche Forschungsgemeinschaft, grant SFB 539 / A4.

Programmer's Niche

Edited by Bill Venables

Save the environment

When you start to learn how to program in S you don't have to get very far into it before you find that the scoping rules can be rather unintuitive. The sort of difficulty that people first encounter is often something like the following (on S-PLUS 2000):

```
> twowaymeans <- function(X, f)
    apply(X, 2, function(x) tapply(x, f, mean))
> twowaymeans(iris[,1:2], iris$Species)
Error in FUN(X): Object "f" not found
Dumped
```

The dismay expressed by disappointed neophytes on S-news is often palpable. This is not helped by the people who point out that on R it does work because of the more natural scoping rules:

```
> twowaymeans(iris[,1:2], iris$Species)
```

```
           Sepal.Length Sepal.Width
setosa            5.006       3.428
versicolor        5.936       2.770
virginica         6.588       2.974
```

Some people even claim to have chosen R over S-PLUS for the sake of the scoping rules alone. Strange, but true.

The different scoping rules are just one consequence of a feature of R, namely that its functions have an "environment" attached to them which is usually called the function closure. In S the search path is the same for all functions, namely the local frame, frame 1, frame 0 and the sequence of data directories or attached objects. This can be altered on the fly and functions may be made to look at, and interfere with, non-standard places on the search path but this is usually regarded as a device strictly for the excessively brave or the excessively lucky. For some, even using frames 0 or 1 at all is considered rather reckless.

This is the territory where R and S diverge very markedly indeed. How is R so different? I will leave readers to follow up this interesting story using the standard references. All I will try to do here is give an extended example that I hope motivates that kind of followup study. In fact I intend to give the "other story" that I referred to so tantalizingly in the first Programmer's Niche article, of course.

Function closures

We can think of the function closure as a little scratchpad attached to the function that initially contains some objects on which its definition may depend. This turns out to be a useful place to put other things which the function needs to have, but which you don't want to re-create every time you call the function itself. If you understand the concept of frame 0 (the frame of the session) in S, this is a bit like a local frame 0, but for that function alone.

Subsets revisited with caching

In my last article on profiling I used an example of a recursive function to generate all possible subsets of size r of a fixed set of size n. The result is an $\binom{n}{r} \times r$ matrix whose rows define the subsets. Some time ago I devised a way of speeding this process up in S by storing (or 'caching') partial results in frame 0. Doug Bates then pointed out that it could be done much more cleanly using function closures in R.

Thinking back on the subset calculation, notice that if you have all possible subsets of size r of the integers $1, 2, \ldots, n$ as a matrix then the subsets of any set of size n stored in a vector `v` can be got by giving `v` the elements of this matrix as an index vector and using the result to fill a similar matrix with the chosen elements of `v`. (Oh well, think about it for a bit.)

The way we generate all possible subsets of size r from n involves repeatedly generating all possible subsets of smaller sizes from a smaller sets. What we are going to do now is generate these indexing vectors and store them in the the function closure. The index vector for subsets of size 4 from sets of size 10, say, will be given the non-standard name, `4 10`. (It is no penalty to use non-standard names here since these objects will always be accessed indirectly.) Then when we need to generate a set of subsets we will check to see if the index vector to do it is cached in the environment first. If it is we do the job by a single index computation; if not we first generate the index by a recursive call and then use it.

Actually it is one of those times when the code is easier to read than an explanation of it. However even the code is not all that easy. The result, however, is a further spectacular increase in speed but at the cost of greatly increasing your memory usage. If you have to do this sort of computation repeatedly, though, the advantages of the cached index vectors in the environment are maintained, of course, so the second time round, even for the large number of subsets, the computation is nearly instantaneous (providing you are not hitting a memory limit and repeatedly swapping, swapping, swapping, ...). So the technique is both interesting and potentially important.

The code

To get a function with an explicit environment (and not just the global environment) we are going to do it in the "old way" by writing a function to generate the function itself. OK, here goes:

```
makeSubsets <- function() {

  putenv <- function(name, value)
    assign(name, value,
           envir = environment(Subsets))

  getenv <- function(name)
    get(name, envir = environment(Subsets))

  thereIsNo <- function(name)
    !exists(name, envir = environment(Subsets))

  function(n, r, v = 1:n) {
    v0 <- vector(mode(v), 0)
    if(r < 0 || r > n) stop("incompatible n, r")
    sub <- function(n, r, v) {
      if(r == 0) v0 else
      if(r == n) v[1:n] else {
        if(r > 1) {
          i1 <- paste(n-1, r-1)
          i2 <- paste(n-1, r)
          if(thereIsNo(i1))
            putenv(i1, sub(n-1, r-1, 1:(n-1)))
          if(thereIsNo(i2))
            putenv(i2, sub(n-1, r,   1:(n-1)))
          m1 <- matrix(v[-1][getenv(i1)],
```

```
                    ncol = r-1)
          m2 <- matrix(v[-1][getenv(i2)],
                       ncol = r)
        } else {
          m1 <- NULL
          m2 <- matrix(v[2:n], ncol = 1)
        }
        rbind(cbind(v[1], m1), m2)
      }
    }
    sub(n, r, v)
  }
}

Subsets <- makeSubsets()
```

The local environment will initially contain the small utility functions `getenv`, `putenv` and `thereIsNo` for doing various things with the local environment. Within the function itself the index matrices are referred to by the constructed character strings `i1` and `i2`.

Here are a few little comparisons on my oldish Sun system:

```
## best from last time
> system.time(x <- subsets2(20, 7))
[1] 22.26  3.96 26.29  0.00  0.00
## first time round
> system.time(X <- Subsets(20, 7))
[1] 4.94 0.25 5.38 0.00 0.00
## second time
> system.time(L <- Subsets(20, 7, letters))
[1] 1.90 0.04 2.00 0.00 0.00
```

These times are actually quite variable and depend a lot on what is going on with the machine itself. Note that with the fastest function we devised last time the computation took about 26 seconds total time. With the caching version the same computation took 5.38 seconds total time the first time and only 2 seconds the next time when we did the same computation with a different set.

Compression

There is a final speculative twist to this story that I can't resist throwing in even though its usefulness will be very machine dependent.

It must be clear that storing oodles of very large index vectors in the local environment will incur a memory overhead that might well become a problem. Can we compress the vectors on the fly in any way to cut down on this? The only way I have been able to see how to do this has been to use so-called "run length encoding". Given a vector, `v`, the function `rle` finds the lengths of each run of consecutive identical values and returns a list of two components: one giving the values that are repeated in each run and the other the lengths of the runs. This is the inverse operation to the one performed by `rep`: if we feed those two components back to `rep` we re-create the original vector.

The function `rle` is one of the neatest examples of slick programming around. It is very short and a nice little puzzle to see how it works. Here is a cut-down version of it that returns a value with list names matching the argument names of `rep`:

```
rle <- function (x) {
  n <- length(x)
  y <- x[-1] != x[-n]
  i <- c(which(y), n)
  list(x = x[i], times = diff(c(0, i)))
}
```

The thing you notice about these subset index vectors (when the subsets are generated in this lexicographic order) is that they do have long runs of repeated values. In fact the run-length encoded version is an object typically only about half as big (in bytes) as the object itself. This produces a compression of the memory requirements, but at an increased computational cost again, of course. To incorporate the idea into our `makeSubsets` function we need to include this specialised version of `rle` in the local environment as well and to modify the getting and putting functions to include the encoding and decoding:

```
    putenv <- function(name, value)
      assign(name, rle(value),
             envir = environment(Subsets))

    getenv <- function(name)
      do.call("rep", get(name,
              envir = environment(Subsets)))
```

No modification to the function itself is needed, and this is one advantage of using accessor functions to deal with the local environment, of course.

I think this is an interesting idea, but I have to report that for all the systems I have tried it upon the increased computational penalty pretty well eliminates the gains made by caching. *Quel domage!*

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

Recent Events

by Friedrich Leisch

This new regular columns will report about recent events related to R, S or statistical computing in general. If you have been to a conference where R turned out to be one of the hot topics (or simply part of a nifty software demonstration), please write a few lines and send them to the editors.

DSC 2001

The second workshop on distributed statistical computing took place at the Vienna University of Technology from March 15 to March 17, 2001. As its predecessor in 1999 it turned out to be a highly stimulating event sparking many discussions on the future of statistical computing, which continued even after dinner and didn't stop until the last beers were ordered long after midnight. Of course the workshop program had a slight bias towards R, but speakers also included John Eaton (Octave), Günther Sawitzki (Voyager), David Smith (S-PLUS), Deborah Swayne (GGobi), Antony Unwin (MANET) and many more. Please have a look at the homepage of the workshop at http://www.ci.tuwien.ac.at/Conferences/DSC.html to see who participated and what topics exactly we talked about. Online proceedings are also available as PDF files.

The workshop was followed by two days of open R core meetings, where all interested could participate and discuss future plans for R and S. The two DSC workshops mark the only two occasions where almost the complete R core team has ever met physically, normally we are spread out over 3 continents with completely incompatible time zones and all work on R has to be organized virtually using the Internet.

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