The R Journal

Volume 1/1, May 2009

A peer-reviewed, open-access publication of the R Foundation for Statistical Computing

Contents

The R Journal is a peer-reviewed publication of the R Foundation for Statistical Computing. Communications regarding this publication should be addressed to the editors. All articles are copyrighted by the respective authors. Please send submissions to regular columns to the respective column editor and all other submissions to the editor-in-chief or another member of the editorial board. More detailed submission instructions can be found on the Journal's homepage.

Editorial

by Vince Carey

Early in the morning of January 7, 2009 — very early, from the perspective of readers in Europe and the Americas, as I was in Almaty, Kazakhstan — I noticed a somewhat surprising title on the front web page of the *New York Times*: "Data Analysts Captivated by R's Power". My prior probability that a favorite programming language would find its way to the *Times* front page was rather low, so I navigated to the article to check. There it was: an early version, minus the photographs. Later a few nice pictures were added, and a *Bits Blog* http://bits.blogs.nytimes.com/2009/01/08/r-you-ready-for-r/ created. As of early May, the blog includes 43 entries received from January 8 to March 12 and includes comment from Trevor Hastie clarifying historical roots of R in S, from Alan Zaslavsky reminding readers of John Chambers's ACM award and its role in funding an *American Statistical Association* prize for students' software development, and from numerous others on aspects of various proprietary and non-proprietary solutions to data analytic computing problems. Somewhat more recently, the *Times* blog was back in the fold, describing a SAS-to-R interface (February 16). As a native New Yorker, it is natural for me to be aware, and glad, of the *Times'* coverage of R; consumers of other mainstream media products are invited to notify me of comparable coverage events elsewhere.

As R's impact continues to expand, R News has transitioned to The R Journal. You are now reading the first issue of this new periodical, which continues the tradition of the newsletter in various respects: articles are peer-reviewed, copyright of articles is retained by authors, and the journal is published electronically through an archive of freely downloadable PDF issues, easily found on the R project homepage and on CRAN. My role as Editor-In-Chief of this first issue of the new Journal is purely accidental – the transition has been in development for several years now, with critical guidance and support from the R Foundation, from a number of R Core members, and from previous editors. I am particularly thankful for John Fox's assistance with this transition, and we are all indebted to co-editors Peter Dalgaard and Heather Turner for technical work on the new Journal's computational infrastructure, undertaken on top of already substantial commitments of editorial effort.

While contemplating the impending step into a new publishing vehicle, John Fox conceived a series of special invited articles addressing views of "The Future of R". This issue includes two articles in this projected series. In the first, John Chambers analyzes R into facets that help us to understand the 'richness' and 'messiness' of R's programming model. In the second, Stefan Theußl and Achim Zeileis describe R-Forge, a new infrastructure for contributing and managing source code of packages destined for CRAN. A number of other future-oriented articles have been promised for this series; we expect to issue a special edition of the Journal collecting these when all have arrived.

The peer-reviewed contributions section includes material on programming for diagram construction, HTML generation, probability model elicitation, integration of the multivariate normal density, Hilbert spectrum analysis, microarray study design, parallel computing support, and the predictive modeling markup language. Production of this issue involved voluntary efforts of 22 referees on three continents, who will be acknowledged in the year-end issue.

Vince Carey
Channing Laboratory, Brigham and Women's Hospital
Vince.Carey@R-project.org

Facets of R

Special invited paper on "The Future of R"

by John M. Chambers

We are seeing today a widespread, and welcome, tendency for non-computer-specialists among statisticians and others to write collections of R functions that organize and communicate their work. Along with the flood of software sometimes comes an attitude that one need only learn, or teach, a sort of basic how-to-write-the-function level of R programming, beyond which most of the detail is unimportant or can be absorbed without much discussion. As delusions go, this one is not very objectionable if it encourages participation. Nevertheless, a delusion it is. In fact, functions are only one of a variety of important facets that R has acquired by intent or circumstance during the three-plus decades of the history of the software and of its predecessor S. To create valuable and trustworthy software using R often requires an understanding of some of these facets and their interrelations. This paper identifies six facets, discussing where they came from, how they support or conflict with each other, and what implications they have for the future of programming with R.

Facets

Any software system that has endured and retained a reasonably happy user community will likely have some distinguishing characteristics that form its style. The characteristics of different systems are usually not totally unrelated, at least among systems serving roughly similar goals—in computing as in other fields, the set of truly distinct concepts is not that large. But in the mix of different characteristics and in the details of how they work together lies much of the flavor of a particular system.

Understanding such characteristics, including a bit of the historical background, can be helpful in making better use of the software. It can also guide thinking about directions for future improvements of the system itself.

The R software and the S software that preceded it reflect a rather large range of characteristics, resulting in software that might be termed rich or messy according to one's taste. Since R has become a very widely used environment for applications and research in data analysis, understanding something of these characteristics may be helpful to the community.

This paper considers six characteristics, which we will call *facets*. They characterize R as:

1. *an interface* to computational procedures of many kinds;
2. *interactive*, hands-on in real time;
3. *functional* in its model of programming;
4. *object-oriented*, "everything is an object";
5. *modular*, built from standardized pieces; and,
6. *collaborative*, a world-wide, open-source effort.

None of these facets is original to R, but while many systems emphasize one or two of them, R continues to reflect them all, resulting in a programming model that is indeed rich but sometimes messy.

The thousands of R packages available from CRAN, BioConductor, R-Forge, and other repositories, the uncounted other software contributions from groups and individuals, and the many citations in the scientific literature all testify to the useful computations created within the R model. Understanding how the various facets arose and how they can work together may help us improve and extend that software.

We introduce the facets more or less chronologically, in three pairs that entered into the software during successive intervals of roughly a decade. To provide some context, here is a brief chronology, with some references. The S project began in our statistics research group at Bell Labs in 1976, evolved into a generally licensed system through the 1980s and continues in the S+ software, currently owned by TIBCO Software Inc. The history of S is summarized in the Appendix to Chambers (2008); the standard books introducing still-relevant versions of S include Becker et al. (1988) (no longer in print), Chambers and Hastie (1992), and Chambers (1998). R was announced to the world in Ihaka and Gentleman (1996) and evolved fairly soon to be developed and managed by the R-core group and other contributors. Its history is harder to pin down, partly because the story is very much still happening and partly because, as a collaborative open-source project, individual responsibility is sometimes hard to attribute; in addition, not all the important new ideas have been described separately by their authors. The `http://www.r-project.org` site points to the on-line manuals, as well as a list of over 70 related books and other material. The manuals are invaluable as a technical resource, but short on background information. Articles in *R News*, the predecessor of this journal, give descriptions of some of the key contributions, such as namespaces (Tierney, 2003), and internationalization (Ripley, 2005).

An interactive interface

The goals for the first version of S in 1976 already combined two facets of computing usually kept

apart, interactive computing and the development of procedural software for scientific applications.

One goal was a better interface to new computational procedures for scientific data analysis, usually implemented then as Fortran subroutines. Bell Labs statistics research had collected and written a variety of these while at the same time numerical libraries and published algorithm collections were establishing many of the essential computational techniques. These procedures were the essence of the actual data analysis; the initial goal was essentially a better interface to procedural computation. A graphic from the first plans for S (Chambers, 2008, Appendix, page 476) illustrated this.

But that interface was to be *interactive*, specifically via a language used by the data analyst communicating in real time with the software. At this time, a few systems had pointed the way for interactive scientific computing, but most of them were highly specialized with limited programming facilities; that is, a limited ability for the user to express novel computations. The most striking exception was APL, created by Kenneth Iverson (1962), an operator-based interactive language with heavy emphasis on general multiway arrays. APL introduced conventions that now seem obvious; for example, the result of evaluating a user's expression was either an assignment or a printed display of the computed object (rather than expecting users to type a print command). But APL at the time had no notion of an interface to procedures, which along with a limited, if exotic, syntax caused us to reject it, although S adopted a number of features from it, such as its approach to general multidimensional arrays.

From its first implementation, the S software combined these two facets: users carried out data analysis interactively by writing expressions in the S language, but much of the programming to extend the system was via new Fortran subroutines accompanied by interfaces to call them from S. The interfaces were written in a special language that was compiled into Fortran.

The result was already a mixed system that had much of the tension between human and machine efficiency that still stimulates debate among R users and programmers. At the same time, the flexibility from that same mixture helped the software to grow and adapt with a growing user community.

Later versions of the system replaced the interface language with individual functions providing interfaces to C or Fortran, but the combination of an interactive language and an interface to compiled procedures remains a key feature of R. Increasingly, interfaces to other languages and systems have been added as well, for example to database systems, spreadsheets and user interface software.

Functional and object-oriented programming

During the 1980s the topics of *functional programming* and *object-oriented programming* stimulated growing interest in the computer science literature. At the same time, I was exploring possibilities for the next S, perhaps "after S". These ideas were eventually blended with other work to produce the "new S", later called Version 3 of S, or S3, (Becker et al., 1988).

As before, user expressions were still interpreted, but were now parsed into objects representing the language elements, mainly function calls. The functions were also objects, the result of parsing expressions in the language that defined the function. As the motto expressed it, "Everything is an object". The procedural interface facet was not abandoned, however, but from the user's perspective it was now defined by functions that provided an interface to a C or Fortran subroutine.

The new programming model was functional in two senses. It largely followed the functional programming paradigm, which defined programming as the evaluation of function calls, with the value uniquely determined by the objects supplied as arguments and with no external side-effects. Not all functions followed this strictly, however.

The new version was functional also in the sense that nearly everything that happened in evaluation was a function call. Evaluation in R is even more functional in that language components such as assignment and loops are formally function calls internally, reflecting in part the influence of Lisp on the initial implementation of R.

Subsequently, the programing model was extended to include classes of objects and methods that specialized functions according to the classes of their arguments. That extension itself came in two stages. First, a thin layer of additional code implemented classes and methods by two simple mechanisms: objects could have an attribute defining their class as a character string or a vector of multiple strings; and an explicitly invoked method dispatch function would match the class of the function's first argument against methods, which were simply saved function objects with a class string appended to the object's name. In the second stage, the version of the system known as S4 provided formal class definitions, stored as a special metadata object, and similarly formal method definitions associated with *generic* functions, also defined via object classes (Chambers, 2008, Chapters 9 and 10, for the R version).

The functional and object-oriented facets were combined in S and R, but they appear more frequently in separate, incompatible languages. Typical object-oriented languages such as C++ or Java are not functional; instead, their programming model is of methods associated with a class rather than with a

function and invoked on an object. Because the object is treated as a reference, the methods can usually modify the object, again quite a different model from that leading to R. The functional use of methods is more complicated, but richer and indeed necessary to support the essential role of functions. A few other languages do combine functional structure with methods, such as Dylan, (Shalit, 1996, chapters 5 and 6), and comparisons have proved useful for R, but came after the initial design was in place.

Modular design and collaborative support

Our third pair of facets arrives with R. The paper by Ihaka and Gentleman (1996) introduced a statistical software system, characterized later as "not unlike S", but distinguished by some ideas intended to improve on the earlier system. A facet of R related to some of those ideas and to important later developments is its approach to *modularity*. In a number of fields, including architecture and computer science, modules are units designed to be useful in themselves and to fit together easily to create a desired result, such as a building or a software system. R has two very important levels of modules: functions and packages.

Functions are an obvious modular unit, especially functions as objects. R extended the modularity of functions by providing them with an *environment*, usually the environment in which the function object was created. Objects assigned in this environment can be accessed by name in a call to the function, and even modified, by stepping outside the strict functional programming model. Functions sharing an environment can thus be used together as a unit. The programming technique resulting is usually called programming with closures in R; for a discussion and comparison with other techniques, see (Chambers, 2008, section 5.4). Because the evaluator searches for external objects in the function's environment, dependencies on external objects can be controlled through that environment. The namespace mechanism (Tierney, 2003), an important contribution to trustworthy programming with R, uses this technique to help ensure consistent behavior of functions in a package.

The larger module introduced by R is probably the most important for the system's growing use, the package. As it has evolved, and continues to evolve, an R package is a module combining in most cases R functions, their documentation, and possibly data objects and/or code in other languages.

While S always had the idea of collections of software for distribution, somewhat formalized in S4 as chapters, the R package greatly extends and improves the ability to share computing results as effective modules. The mechanism benefits from some essential tools provided in R to create, develop, test, and distribute packages. Among many benefits for users, these tools help programmers create software that can be installed and used on the three main operating systems for computing with data—Windows, Linux, and Mac OS X.

The sixth facet for our discussion is that R is a *collaborative* enterprise, which a large group of people share in and contribute to. Central to the collaborative facet is, of course, that R is a freely available, open-source system, both the core R software and most of the packages built upon it. As most readers of this journal will be aware, the combination of the collaborative efforts and the package mechanism have enormously increased the availability of techniques for data analysis, especially the results of new research in statistics and its applications. The package management tools, in fact, illustrate the essential role of collaboration. Another highly collaborative contribution has facilitated use of R internationally (Ripley, 2005), both by the use of locales in computations and by the contribution of translations for R documentation and messages.

While the collaborative facet of R is the least technical, it may eventually have the most profound implications. The growth of the collaborative community involved is unprecedented. For example, a plot in Fox (2008) shows that the number of packages in the CRAN repository exhibited literal exponential growth from 2001 to 2008. The current size of this and other repositories implies at a conservative estimate that several hundreds of contributors have mastered quite a bit of technical expertise and satisfied some considerable formal requirements to offer their efforts freely to the worldwide scientific community.

This facet does have some technical implications. One is that, being an open-source system, R is generally able to incorporate other open-source software and does indeed do so in many areas. In contrast, proprietary systems are more likely to build extensions internally, either to maintain competitive advantage or to avoid the free distribution requirements of some open-source licenses. Looking for quality open-source software to extend the system should continue to be a favored strategy for R development.

On the downside, a large collaborative enterprise with a general practice of making collective decisions has a natural tendency towards conservatism. Radical changes do threaten to break currently working features. The future benefits they might bring will often not be sufficiently persuasive. The very success of R, combined with its collaborative facet, poses a challenge to cultivate the "next big step" in software for data analysis.

Implications

A number of specific implications of the facets of R, and of their interaction, have been noted in previous sections. Further implications are suggested in considering the current state of R and its future possibilities.

Many reasons can be suggested for R's increasing popularity. Certainly part of the picture is a productive interaction among the interface facet, the modularity of packages, and the collaborative, open-source approach, all in an interactive mode. From the very beginning, S was not viewed as a set of procedures (the model for SAS, for example) but as an interactive system to support the implementation of new procedures. The growth in packages noted previously reflects the same view.

The emphasis on interfaces has never diminished and the range of possible procedures across the interface has expanded greatly. That R's growth has been especially strong in academic and other research environments is at least partly due to the ability to write interfaces to existing and new software of many forms. The evolution of the package as an effective module and of the repositories for contributed software from the community have resulted in many new interfaces.

The facets also have implications for the possible future of R. The hope is that R can continue to support the implementation and communication of important techniques for data analysis, contributed and used by a growing community. At the same time, those interested in new directions for statistical computing will hope that R or some future alternative engages a variety of challenges for computing with data. The collaborative community seems the only likely generator of the new features required but, as noted in the previous section, combining the new with maintenance of a popular current collaborative system will not be easy.

The encouragement for new packages generally and for interfaces to other software in particular are relevant for this question also. Changes that might be difficult if applied internally to the core R implementation can often be supplied, or at least approximated, by packages. Two examples of enhancements considered in discussions are an optional object model using mutable references and computations using multiple threads. In both cases, it can be argued that adding the facility to R could extend its applicability and performance. But a substantial programming effort and knowledge of the current implementation would be required to modify the internal object model or to make the core code thread-safe. Issues of back compatibility are likely to arise as well. Meanwhile, less ambitious approaches in the form of packages that approximate the desired behavior have been written. For the second example especially, these may interface to other software designed to provide this facility.

Opinions may reasonably differ on whether these "workarounds" are a good thing or not. One may feel that the total effort devoted to multiple approximate solutions would have been more productive if coordinated and applied to improving the core implementation. In practice, however, the nature of the collaborative effort supporting R makes it much easier to obtain a modest effort from one or a few individuals than to organize and execute a larger project. Perhaps the growth of R will encourage the community to push for such projects, with a new view of funding and organization. Meanwhile, we can hope that the growing communication facilities in the community will assess and improve on the existing efforts. Particularly helpful facilities in this context include the CRAN task views (http://cran.r-project.org/web/views/), the mailing lists and publications such as this journal.

Bibliography

R. A. Becker, J. M. Chambers, and A. R. Wilks. *The New S Language*. Chapman & Hall, London, 1988.

J. M. Chambers. *Programming with Data*. Springer, New York, 1998. URL http://cm.bell-labs.com/cm/ms/departments/sia/Sbook/. ISBN 0-387-98503-4.

J. M. Chambers. *Software for Data Analysis: Programming with R*. Springer, New York, 2008. ISBN 978-0-387-75935-7.

J. M. Chambers and T. J. Hastie. *Statistical Models in S*. Chapman & Hall, London, 1992. ISBN 9780412830402.

J. Fox. Editorial. *R News*, 8(2):1–2, October 2008. URL http://CRAN.R-project.org/doc/Rnews/.

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

K. E. Iverson. *A Programming Language*. Wiley, 1962.

B. D. Ripley. Internationalization features of R 2.1.0. *R News*, 5(1):2–7, May 2005. URL http://CRAN.R-project.org/doc/Rnews/.

A. Shalit. *The Dylan Reference Manual*. Addison-Wesley Developers Press, Reading, Mass., 1996. ISBN 0-201-44211-6.

L. Tierney. Name space management for R. *R News*, 3(1):2–6, June 2003. URL http://CRAN.R-project.org/doc/Rnews/.

John M. Chambers
Department of Statistics
Stanford University
USA
jmc@r-project.org

Collaborative Software Development Using R-Forge

Special invited paper on "The Future of R"

by Stefan Theußl and Achim Zeileis

Introduction

Open source software (OSS) is typically created in a decentralized self-organizing process by a community of developers having the same or similar interests (see the famous essay by Raymond, 1999). A key factor for the success of OSS over the last two decades is the Internet: Developers who rarely meet face-to-face can employ new means of communication, both for rapidly writing and deploying software (in the spirit of Linus Torvald's "release early, release often paradigm"). Therefore, many tools emerged that assist a collaborative software development process, including in particular tools for source code management (SCM) and version control.

In the R world, SCM is not a new idea; in fact, the R Development Core Team has always been using SCM tools for the R sources, first by means of Concurrent Versions System (CVS, see Cederqvist et al., 2006), and then via Subversion (SVN, see Pilato et al., 2004). A central repository is hosted by ETH Zürich mainly for managing the development of the base R system. Mailing lists like R-help, R-devel and many others are currently the main communication channels in the R community.

Also beyond the base system, many R contributors employ SCM tools for managing their R packages, e.g., via web-based SVN repositories like SourceForge (`http://SourceForge.net/`) or Google Code (`http://Code.Google.com/`). However, there has been no central SCM repository providing services suited to the specific needs of R package developers. Since early 2007, the R-project offers such a central platform to the R community. R-Forge (`http://R-Forge.R-project.org/`) provides a set of tools for source code management and various web-based features. It aims to provide a platform for collaborative development of R packages, R-related software or further projects. R-Forge is closely related to the most famous of such platforms—the world's largest OSS development website—namely `http://SourceForge.net/`.

The remainder of this article is organized as follows. First, we present the core features that R-Forge offers to the R community. Second, we give a hands-on tutorial on how users and developers can get started with R-Forge. In particular, we illustrate how people can register, set up new projects, use R-Forge's SCM facilities, provide their packages on R-Forge, host a project-specific website, and how package maintainers submit a package to the Comprehensive R Archive Network (CRAN, `http://CRAN.R-project.org/`). Finally, we summarize recent developments and give a brief outlook to future work.

R-Forge

R-Forge offers a central platform for the development of R packages, R-related software and other projects.

R-Forge is based on GForge (Copeland et al., 2006) which is an open source fork of the 2.61 SourceForge code maintained by Tim Perdue, one of the original SourceForge authors. GForge has been modified to provide additional features for the R community, namely a CRAN-style repository for hosting development releases of R packages as well as a quality management system similar to that of CRAN. Packages hosted on R-Forge are provided in source form as well as in binary form for Mac OS X and Windows. They can be downloaded from the website of the corresponding project on R-Forge or installed directly in R; for a package **foo**, say, `install.packages("foo", repos = "http://R-Forge.R-project.org")`.

On R-Forge, developers organize their work in so-called "Projects". Every project has various tools and web-based features for software development, communcation and other services. All features mentioned in the following sections are accessible via so-called "Tabs": e.g., user accounts can be managed in the *My Page* tab or a list of available projects can be displayed using the *Project Tree* tab.

Since starting the platform in early 2007, more and more interested users registered their projects on R-Forge. Now, after more than two years of development and testing, around 350 projects and more than 900 users are registered on R-Forge. This and the steadily growing list of feature requests show that there is a high demand for centralized source code management tools and for releasing prototype code frequently among the R community.

In the next three sections, we summarize the core features of R-Forge and what R-Forge offers to the R community in terms of collaborative development of R-related software projects.

Source code management

When carrying out software projects, source files change over time, new files get created and old files deleted. Typically, several authors work on several computers on the same and/or different files and

keeping track of every change can become a tedious task. In the open source community, the general solution to this problem is to use version control, typically provided by the majority of SCM tools. For this reason R-Forge utilizes SVN to facilitate the developer's work when creating software.

A central repository ensures that the developer always has access to the current version of the project's source code. Any of the authorized collaborators can "check out" (i.e., download) or "update" the project file structure, make the necessary changes or additions, delete files from the current revision and finally "commit" changes or additions to the repository. More than that, SVN keeps track of the complete history of the project file structure. At any point in the development stage it is possible to go back to any previous stage in the history to inspect and restore old files. This is called version control, as every stage automatically is assigned a unique version number which increases over time.

On R-Forge such a version-controlled repository is automatically created for each project. To get started, the project members just have to install the client of their choice (e.g., Tortoise SVN on Windows or svnX on Mac OS X) and check out the repository. In addition to the inherent backup of every version within the repository a backup of the whole repository is generated daily.

A rights management system assures that, by default, anonymous users have read access and developers have write access to the data associated with a certain project on R-Forge. More precisely, registered users can be granted one of several roles: e.g., the "Administrator" has all rights including the right to add new users to the project or release packages directly to CRAN. He/she is usually the package maintainer, the project leader or has registered the project originally. Other members of a project typically have either the role "Senior Developer" or "Junior Developer" which both have permission to commit to the project SVN repository and examine the log files in the *R Packages* tab (the differences between the two roles are subtle, e.g., senior developers additionally have administrative privileges in several places in the project). When we speak of developers in subsequent sections we refer to project members having the rights at least of a junior developer.

Release and quality management

Development versions of a software project are typically prototypes and are subject to many changes. Thus, R-Forge offers two tools which assist the developers in improving the quality of their source code.

First, it offers a quality management system similar to that of CRAN. Packages on R-Forge are checked in a standardized way on different platforms based on `R CMD check` at least once daily. The resulting log files can be examined by the project developers so that they can improve the package to pass all tests on R-Forge and subsequently on CRAN.

Second, bug tracking systems allow users to notify package authors about problems they encounter. In the spirit of OSS—given enough eyeballs, all bugs are shallow (Raymond, 1999)—peer code review leads to an overall improvement of the quality of software projects.

Additional features

A number of further tools, of increased interest for larger projects, help developers to coordinate their work and to communicate with their user base. These tools include:

- Project websites: Developers may present their work on a subdomain of R-Forge, e.g., `http://foo.R-Forge.R-project.org/`, or via a link to an external website.
- Mailing lists: By default a list `foo-commits@lists.R-Forge.R-project.org` is automatically created for each project. Additional mailing lists can be set up easily.
- Project categorization: Administrators may categorize their project in several so-called "Trove Categories" in the *Admin* tab of their project (under *Trove Categorization*). For each category three different items can be selected. This enables users to quickly find what they are looking for using the *Project Tree* tab.
- News: Announcements and other information related to a project can be put on the project summary page as well as on the home page of R-Forge. The latter needs approval by one of the R-Forge administrators. All items are available as RSS feeds.
- Forums: Discussion forums can be set up separately by the project administrators.

How to get started

This section is intended to be a hands-on tutorial for new users. Depending on familiarity with the systems/tools involved the instructions might be somewhat brief. In case of problems we recommend consulting the user's manual (R-Forge Administration and Development Team, 2008) which contains detailed step-by-step instructions.

When accessing the URL `http://R-Forge.R-project.org/` the home page is presented (see Figure 1). Here one can

- login,
- register a user or a project,

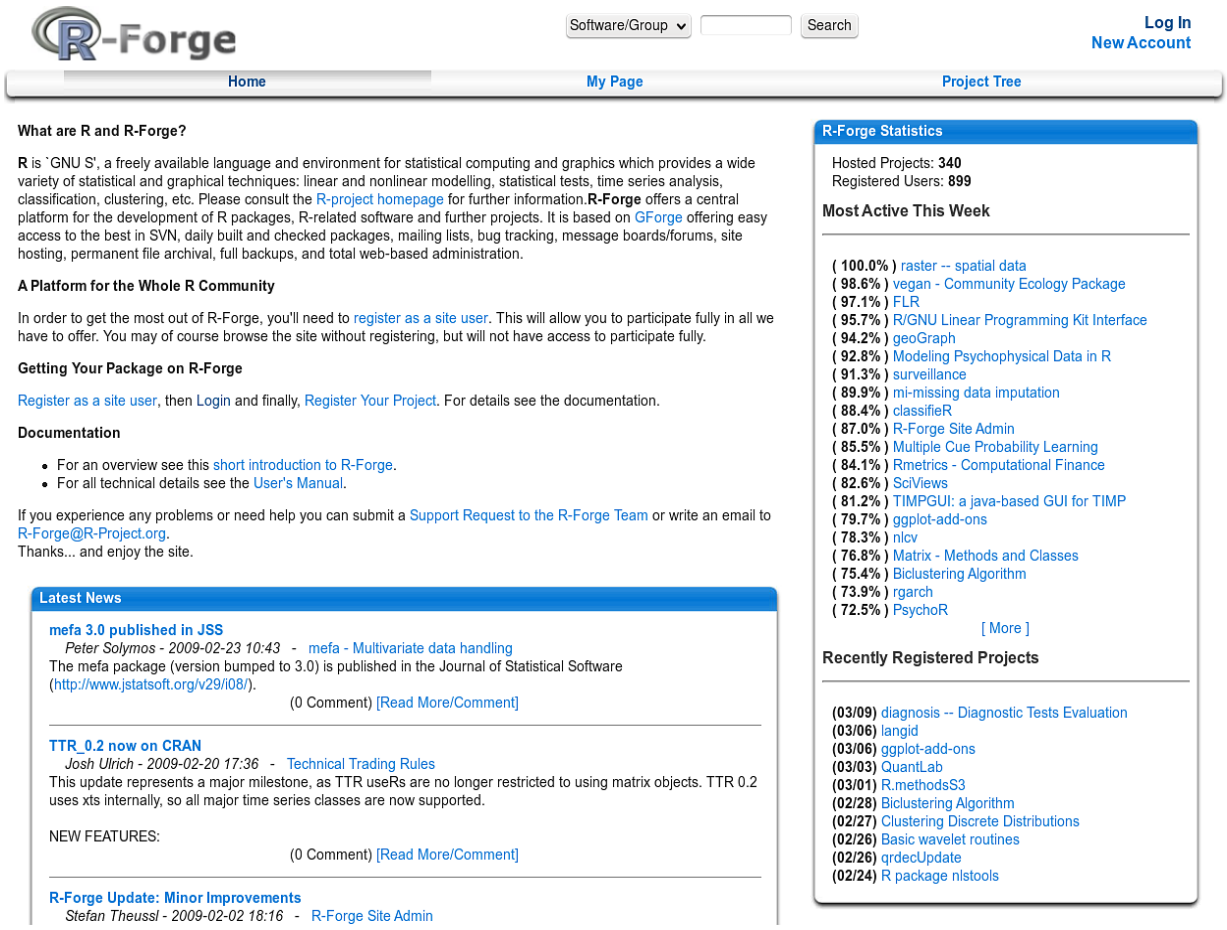


Figure 1: Home page of R-Forge on 2009-03-10

- download the documentation,
- examine the latest news and changes,
- go to a specific project website either by searching for available projects (top middle of the page), by clicking on one of the projects listed on the right, or by going through the listing in the *Project Tree* tab.

Registered users can access their personal page via the tab named *My Page*. Figure 2 shows the personal page of the first author.

Registering as a new user

To use R-Forge as a developer, one has to register as a site user. A link on the main web site called *New Account* on the top right of the home page leads to the corresponding registration form.

After submitting the completed form, an e-mail is sent to the given address containing a link for activating the account. Subsequently, all R-Forge features, including joining existing or creating new projects, are available to logged-in users.

Registering a project

There are two possibilities to register a project: Clicking on *Register Your Project* on the home page or going to the *My Page* tab and clicking on *Register Project* (see Figure 2 below the main tabs). Both links lead to a form which has to be filled out in order to finish the registration process. In the text field "Project Public Description" the registrant is supposed to enter a concise description of the project. This text and the "Project Full Name" will be shown in several places on R-Forge (see e.g., Figure 1). The text entered in the field "Project Purpose And Summarization" is additional information for the R-Forge administrators, inspected for approval. "Project Unix Name" refers to the name which uniquely determines the project. In the case of a project that contains a single R package, the project Unix name typically corresponds to the package name (in its lower-case version). Restrictions according to the Unix file system convention force Unix names to be in lower case (and will be converted automatically if they are typed in upper case).

After the completed form is submitted, the project has to be approved by the R-Forge administrators and a confirmation e-mail is sent to the registrant upon approval. After that, the regis-

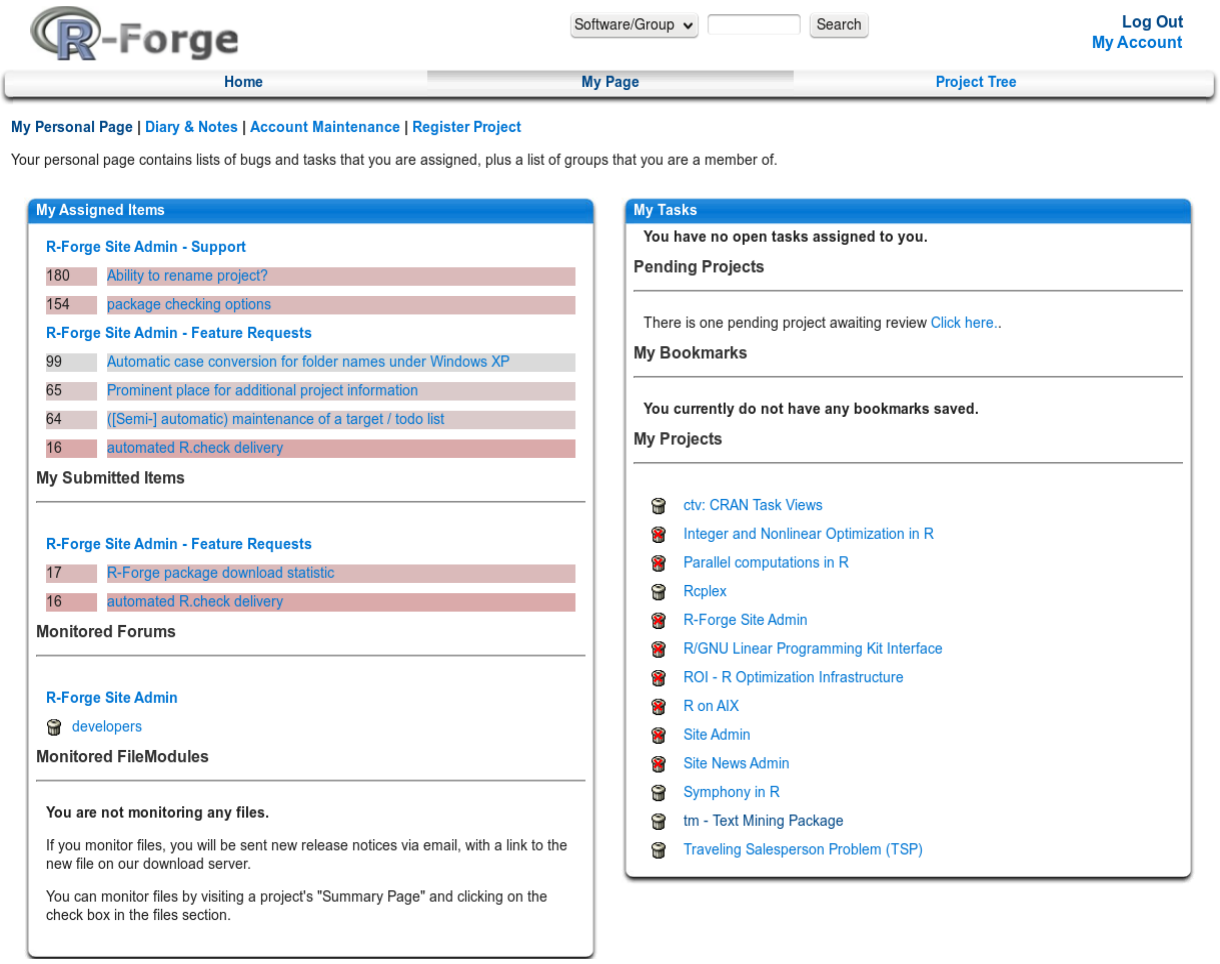


Figure 2: The *My Page* tab of the first author

trant automatically becomes the project administrator and the standardized web area of the project (`http://R-Forge.R-project.org/projects/foo/`) is immediately available on R-Forge. This web area includes a *Summary* page, an *Admin* tab visible only to the project administrators, and various other tabs depending on the features enabled for this project in the *Admin* tab. To present the new project to a broader community the name of the project additionally is promoted on the home page under "Recently Registered Projects" (see Figure 1).

Furthermore, within an hour after approval a default mailing list and the project's SVN repository containing a 'README' file and two pre-defined directories called 'pkg' and 'www' are created. The content of these directories is used by the R-Forge server for creating R packages and a project website, respectively.

SCM and R packages

The first step after creation of a project is typically to start generation of content for one (or more) R package(s) in the 'pkg' directory. Developers can either start committing changes via SVN as usual or—if the package is already version-controlled somewhere else—the corresponding parts of the repository including the history can be migrated to R-Forge (see Section 6 of the user's manual).

The *SCM* tab of a project explains how the corresponding SVN repository located at `svn://svn.R-Forge.R-project.org/svnroot/foo` can be checked out. From this URL the sources are checked out either anonymously without write permission (enabled by default) or as developer using an encrypted authentication method, namely secure shell (`ssh`). Via this secure protocol (`svn+ssh://` followed by the registered user name) developers have full access to the repository. Typically, the developer is authenticated via the registered password but it is also possible to upload a secure shell key (updated hourly) to make use of public/private key authentication. Section 3.2 of the user's manual explains this process in more detail.

To make use of the package builds and checks the package source code has to be put into the 'pkg' directory of the repository (i.e., 'pkg/DESCRIPTION', 'pkg/R', 'pkg/man', etc.) or, alternatively, a subdirectory of 'pkg'. The latter structure allows developers to have more than one package in a single project; e.g.,

if a project consists of the packages `foo` and `bar`, then the source code is located in 'pkg/foo' and 'pkg/bar', respectively.

R-Forge automatically examines the 'pkg' directory of every repository and builds the package sources as well as the package binaries on a daily basis for Mac OS X and Windows (if applicable). The package builds are provided in the *R Packages* tab (see Figure 3) for download or can be installed directly in R using `install.packages("foo", repos="http://R-Forge.R-project.org")`. Furthermore, in the *R Packages* tab developers can examine logs of the build and check process on different platforms.

To release a package to CRAN the project administrator clicks on *Submit this package to CRAN* in the *R Packages* tab. Upon confirmation, a message will be sent to `CRAN@R-project.org` and the latest successfully built source package (the '.tar.gz' file) is automatically copied to `ftp://CRAN.R-project.org/incoming/`. Note that packages are built once daily, i.e., the latest source package does not include more recent code committed to the SVN repository.

Figure 3 shows the *R Packages* tab of the **tm** project (`http://R-Forge.R-project.org/projects/tm/`) as one of the project administrators would see it. Depending on whether you are a member of the project or not and your rights you will see only parts of the information provided for each package.

Further steps

A customized project website, accessible through `http://foo.R-Forge.R-project.org/` where `foo` corresponds to the unix name of the project, is managed via the 'www' directory. The website gets updated every hour.

The changes made to the project can be examined by entering the corresponding standardized web area. On entry, the *Summary* page is shown. Here, one can

- examine the details of the project including a short description and a listing of the administrators and developers,
- follow a link leading to the project homepage,
- examine the latest news announcements (if available),
- go to other sections of the project like *Forums*, *Tracker*, *Lists*, *R Packages*, etc.
- follow the download link leading directly to the available packages of the project (i.e., the *R Packages* tab).

Furthermore, meta-information about a project can be supplied in the *Admin* tab via so-called "Trove Categorization".

Recent and future developments

In this section, we briefly summarize the major changes and updates to the R-Forge system during the last few months and give an outlook to future developments.

Recently added features and major changes include:

- New defaults for freshly registered projects: Only the tabs *Lists*, *SCM* and *R packages* are enabled initially. *Forums*, *Tracker* and *News* can be activated separately (in order not to overwhelm new users with too many features) using *Edit Public Info* in the *Admin* tab of the project. Experienced users can decide which features they want and activate them.
- An enhanced structure in the SVN repository allowing multiple packages in a single project (see above).
- The R package **RForgeTools** (Theußl, 2009) contains platform-independent package building and quality management code used on the R-Forge servers.
- A modified *News* submit page offering two types of submissions: project-specific and global news. The latter needs approval by one of the R-Forge administrators (default: project-only submission).
- Circulation of SVN commit messages can be enabled in the *SCM Admin* tab by project administrators. The mailing list mentioned in the text field is used for delivering the SVN commit messages (default: off).
- Mailing list search facilities provided by the Swish-e engine which can be accessed via the *List* tab (private lists are not included in the search index).

Further features and improvements which are currently on the wishlist or under development include

- a Wiki,
- task management facilities,
- a re-organized tracker more compatible with R package development and,
- an update of the underlying GForge system to its successor FusionForge (`http://FusionForge.org/`).

For suggestions, problems, feature requests, and other questions regarding R-Forge please contact `R-Forge@R-project.org`.

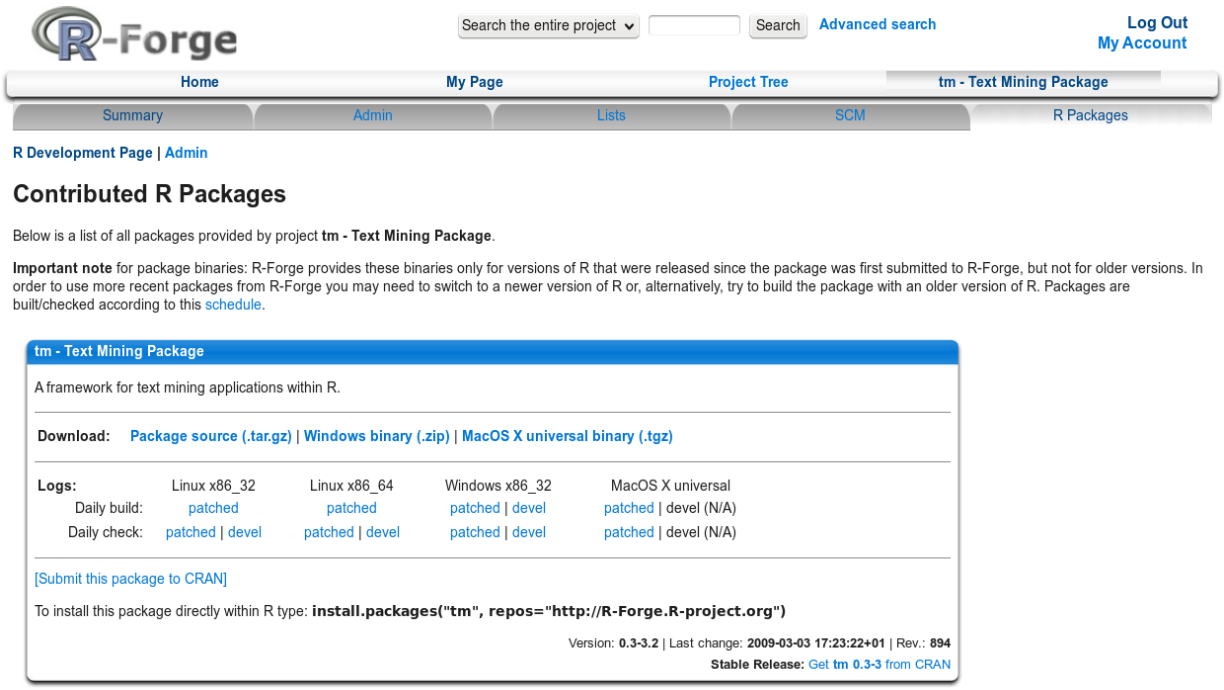


Figure 3: *R Packages* tab of the **tm** project

Acknowledgments

Setting up this project would not have been possible without Douglas Bates and the University of Wisconsin as they provided us with a server for hosting this platform. Furthermore, the authors would like to thank the Computer Center of the Wirtschaftsuniversität Wien for their support and for providing us with additional hardware as well as a professional server infrastructure.

Bibliography

P. Cederqvist et al. *Version Management with CVS*. Network Theory Limited, Bristol, 2006. Full book available online at http://ximbiot.com/cvs/manual/.

T. Copeland, R. Mas, K. McCullagh, T. Perdue, G. Smet, and R. Spisser. *GForge Manual*, 2006. URL http://gforge.org/gf/download/docmanfileversion/8/1700/gforge_manual.pdf.

C. M. Pilato, B. Collins-Sussman, and B. W. Fitzpatrick. *Version Control with Subversion*. O'Reilly, 2004. Full book available online at http://svnbook.red-bean.com/.

R-Forge Administration and Development Team. *R-Forge User's Manual*, 2008. URL http://download.R-Forge.R-project.org/R-Forge.pdf.

E. S. Raymond. The Cathedral and the Bazaar. *Knowledge, Technology, and Policy*, 12(3):23–49, 1999.

S. Theußl. *RForgeTools: R-Forge Build and Check Tools*, 2009. URL http://R-Forge.R-project.org/projects/site/. R package version 0.3-3.

Stefan Theußl
Department of Statistics and Mathematics
WU Wirtschaftsuniversität Wien
Austria
stefan.theussl@wu.ac.at

Achim Zeileis
Department of Statistics and Mathematics
WU Wirtschaftsuniversität Wien
Austria
achim.zeileis@wu.ac.at

Drawing Diagrams with R

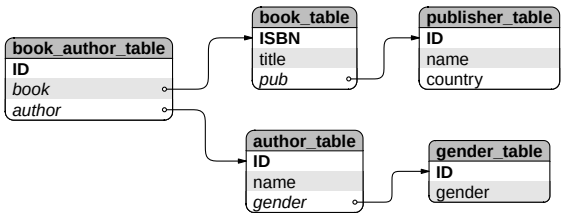
by Paul Murrell

R provides a number of well-known *high-level* facilities for producing sophisticated statistical plots, including the "traditional" plots in the **graphics** package (R Development Core Team, 2008), the Trellis-style plots provided by **lattice** (Sarkar, 2008), and the grammar-of-graphics-inspired approach of **ggplot2** (Wickham, 2009).

However, R also provides a powerful set of *low-level* graphics facilities for drawing basic shapes and, more importantly, for arranging those shapes relative to each other, which can be used to draw a wide variety of graphical images. This article highlights some of R's low-level graphics facilities by demonstrating their use in the production of *diagrams*. In particular, the focus will be on some of the useful things that can be done with the low-level facilities provided by the **grid** graphics package (Murrell, 2002, 2005b,a).

Starting at the end

An example of the type of diagram that we are going to work towards is shown below. We have several "boxes" that describe table schema for a database, with lines and arrows between the boxes to show the relationships between tables.



To forestall some possible misunderstandings, the sort of diagram that we are talking about is one that is designed *by hand*. This is not a diagram that has been automatically laid out.

The sort of diagram being addressed is one where the author of the diagram has a clear idea of what the end result will roughly look like—the sort of diagram that can be sketched with pen and paper. The task is to produce a pre-planned design, using a computer to get a nice crisp result.

That being said, a reasonable question is "why not draw it by hand?", for example, using a free-hand drawing program such as **Dia** (Larsson, 2008). The advantage of using R code to produce this sort of image is that code is easier to reproduce, reuse, maintain, and fine-tune with accuracy. The thought of creating this sort of diagram by pushing objects around the screen with a mouse fills me with dread. Maybe I'm just not a very GUI guy.

Before we look at drawing diagrams with the core R graphics facilties, it is important to acknowledge that several contributed R packages already provide facilities for drawing diagrams. The **Rgraphviz** (Gentry et al., 2008) and **igraph** (Csardi and Nepusz, 2006) packages provide automated layout of node-and-edge graphs, and the **shape** and **diagram** packages (Soetaert, 2008b,a) provide functions for drawing nodes of various shapes with lines and arrows between them, with manual control over layout.

In this article, we will only be concerned with drawing diagrams with a small number of elements, so we do not need the automated layout facilities of **Rgraphviz** or **igraph**. Furthermore, while the **shape** and **diagram** packages provide flexible tools for building node-and-edge diagrams, the point of this article is to demonstrate low-level **grid** functions. We will use a node-and-edge diagram as the motivation, but the underlying ideas can be applied to a much wider range of applications.

In each of the following sections, we will meet a basic low-level graphical tool and demonstrate how it can be used in the generation of the pieces of an overall diagram, or how the tool can be used to combine pieces together in convenient ways.

Graphical primitives

One of the core low-level facilities of R graphics is the ability to draw basic shapes. The typical *graphical primitives* such as text, circles, lines, and rectangles are all available.

In this case, the shape of each box in our diagram is not quite as simple as a rectangle because it has rounded corners. However, a rounded rectangle is also one of the graphical primitives that the **grid** package provides.[1]

The code below draws a rounded rectangle with a text label in the middle.

```
> library(grid)

> grid.roundrect(width=.25)
> grid.text("ISBN")
```

[1] From R version 2.9.0; prior to that, a simpler rounded rectangle was available via the `grid.roundRect()` function in the **RGraphics** package.

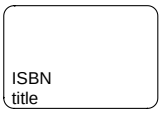
Viewports

A feature of the boxes in the diagram at the beginning of this article is that the text is carefully positioned relative to the rounded rectangle; the text is left-aligned within the rectangle. This careful positioning requires knowing where the left edge of the rectangle is on the page. Calculating those positions is annoyingly tricky and only becomes more annoying if at some later point the position of the box is adjusted and the positions of the text labels have to be calculated all over again.

Using a **grid** viewport makes this sort of positioning very simple. The basic idea is that we can create a viewport where the box is going to be drawn and then do all of our drawing within that viewport. Positioning text at the left edge of a viewport is very straightforward, and if we need to shift the box, we simply shift the viewport and the text automatically tags along for the ride. All of this applies equally to positioning the text vertically within the box.

In the code below, we create a viewport for the overall box, we draw a rounded rectangle occupying the entire viewport, then we draw text 2 mm from the left hand edge of the viewport and 1.5 lines of text up from the bottom of the viewport. A second line of text is also added, 0.5 lines of text from the bottom.

```
> pushViewport(viewport(width=.25))
> grid.roundrect()
> grid.text("ISBN",
            x=unit(2, "mm"),
            y=unit(1.5, "lines"),
            just="left")
> grid.text("title",
            x=unit(2, "mm"),
            y=unit(0.5, "lines"),
            just="left")
> popViewport()
```



Coordinate systems

The positioning of the labels within the viewport in the previous example demonstrates another useful feature of the **grid** graphics system: the fact that locations can be specified in a variety of *coordinate systems* or *units*. In that example, the text was positioned horizontally in terms of millimetres and vertically in terms of lines of text (which is based on the font size in use).

As another example of the use of these different units, we can size the overall viewport so that it is just the right size to fit the text labels. In the following code, the height of the viewport is based on the number of labels and the width of the viewport is based on the width of the largest label, plus a 2 mm gap either side. This code also simplifies the labelling by drawing both labels in a single `grid.text()` call.

```
> labels <- c("ISBN", "title")
> vp <-
    viewport(width=max(stringWidth(labels))+
                   unit(4, "mm"),
             height=unit(length(labels),
                         "lines"))
> pushViewport(vp)
> grid.roundrect()
> grid.text(labels,
            x=unit(2, "mm"),
            y=unit(2:1 - 0.5, "lines"),
            just="left")
> popViewport()
```

Clipping

Another feature of the boxes that we want to produce is that they have shaded backgrounds. Looking closely, there are some relatively complex shapes involved in this shading. For example, the grey background for the "heading" of each box has a curvy top, but a flat bottom. These are not simple rounded rectangles, but some unholy alliance of a rounded rectangle and a normal rectangle.

It is possible, in theory, to achieve any sort of shape with R because there is a general polygon graphical primitive. However, as with the positioning of the text labels, determining the exact boundary of this polygon is not trivial and there are easier ways to work.

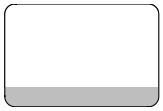
In this case, we can achieve the result we want using *clipping*, so that any drawing that we do is only visible on a restricted portion of the page. R does not provide clipping to arbitrary regions, but it is possible to set the clipping region to any *rectangular* region.

The basic idea is that we will draw the complete rounded rectangle, then set the clipping region for the box viewport so that no drawing can occur in the last line of text in the box and then draw the rounded rectangle again, this time with a different background. If we continue doing this, we end up with bands of different shading.

The following code creates an overall viewport for a box and draws a rounded rectangle with a grey fill. The code then sets the clipping region to start one line of text above the bottom of the viewport and draws another rounded rectangle with a white fill. The effect is to leave just the last line of the original

grey rounded rectangle showing beneath the white rounded rectangle that has had its last line clipped.

```
> pushViewport(viewport(width=.25))
> grid.roundrect(gp=gpar(fill="grey"))
> grid.clip(y=unit(1, "lines"),
            just="bottom")
> grid.roundrect(gp=gpar(fill="white"))
> popViewport()
```



Drawing curves

Another basic shape that is used in the overall diagram is a nice curve from one box to another.

In addition to the basic functions to draw *straight* lines in R, there are functions that draw *curves*. In particular, R provides a graphical primitive called an *X-spline* (Blanc and Schlick, 1995). The idea of an X-spline is that we define a set of *control points* and a curve is drawn either through or near to the control points. Each control point has a parameter that specifies whether to create a sharp corner at the control point, or draw a smooth curve through the control point, or draw a smooth curve that passes nearby.

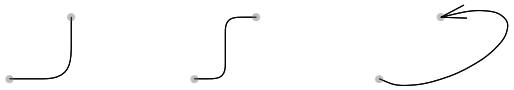
The following code sets up sets of three control points and draws an X-spline relative to each set of control points. The first curve makes a sharp corner at the middle control point, the second curve makes a smooth corner *through* the middle control point, and the third curve makes a smooth corner *near* the middle control point. The control points are drawn as grey dots for reference (code not shown).

```
> x1 <- c(0.1, 0.2, 0.2)
> y1 <- c(0.2, 0.2, 0.8)
> grid.xspline(x1, y1)
> x2 <- c(0.4, 0.5, 0.5)
> y2 <- c(0.2, 0.2, 0.8)
> grid.xspline(x2, y2, shape=-1)
> x3 <- c(0.7, 0.8, 0.8)
> y3 <- c(0.2, 0.2, 0.8)
> grid.xspline(x3, y3, shape=1)
```

Determining where to place the control points for a curve between two boxes is another one of those annoying calculations, so a more convenient option is provided by a *curve* graphical primitive in **grid**. The idea of this primitive is that we simply specify the start and end points of the curve and R figures out a set of reasonable control points to produce an appropriate X-spline. It is also straightforward to add an arrow to either end of any straight or curvy line that R draws.

The following code draws three curves between pairs of end points. The first curve draws the default "city-block" line between end points, with a smooth corner at the turning point, the second curve is similar, but with an extra corner added, and the third curve draws a single wide, smooth corner that is distorted towards the end point. The third curve also has an arrow at the end.

```
> x1a <- 0.1; x1b <- 0.2
> y1a <- 0.2; y1b <- 0.8
> grid.curve(x1a, y1a, x1b, y1b)
> x2a <- 0.4; x2b <- 0.5
> y2a <- 0.2; y2b <- 0.8
> grid.curve(x2a, y2a, x2b, y2b,
             inflect=TRUE)
> x3a <- 0.7; x3b <- 0.8
> y3a <- 0.2; y3b <- 0.8
> grid.curve(x3a, y3a, x3b, y3b,
             ncp=8, angle=135,
             square=FALSE,
             curvature=2,
             arrow=arrow(angle=15))
```



Graphical functions

The use of graphical primitives, viewports, coordinate systems, and clipping, as described so far, can be used to produce a box of the style shown in the diagram at the start of the article. For example, the following code produces a box containing three labels, with background shading to assist in differentiating among the labels.

```
> labels <- c("ISBN", "title", "pub")
> vp <-
    viewport(width=max(stringWidth(
                               labels))+
                     unit(4, "mm"),
             height=unit(length(labels),
                         "lines"))
> pushViewport(vp)
> grid.roundrect()
> grid.clip(y=unit(1, "lines"),
            just="bottom")
> grid.roundrect(gp=gpar(fill="grey"))
> grid.clip(y=unit(2, "lines"),
            just="bottom")
> grid.roundrect(gp=gpar(fill="white"))
> grid.clip()
> grid.text(labels,
            x=unit(rep(2, 3), "mm"),
            y=unit(3:1 - .5, "lines"),
```

```
            just="left")
> popViewport()
```

However, in the sort of diagram that we want to produce, there will be several such boxes. Rather than write separate code for each box, it makes sense to write a general function that will work for any set of labels. Such a function is shown in Figure 1 and the code below uses this function to draw two boxes side by side.

```
> tableBox(c("ISBN", "title", "pub"),
           x=0.3)
> tableBox(c("ID", "name", "country"),
           x=0.7)
```

This function represents the simplest way to efficiently reuse graphics code and to provide graphics code for others to use. However, there are benefits to be gained from going beyond this procedural programming style to a slightly more complicated object-oriented approach.

Graphical objects

In order to achieve the complete diagram introduced at the start of this article, we need one more step: we need to draw lines and arrows from one box to another. We already know how to draw lines and curves between two points; the main difficulty that remains is calculating the exact position of the start and end points, because these locations depend on the locations and dimensions of the boxes. The calculations could be done by hand for each individual curve, but as we have seen before, there are easier ways to work. The crucial idea for this step is that we want to create not just a graphical function that encapsulates how to draw a box, but define a *graphical object* that encapsulates information about a box.

The code in Figure 2 defines such a graphical object, plus a few other things that we will get to shortly. The first thing to concentrate on is the `boxGrob()` function. This function creates a `"box"` graphical object. In order to do this, all it has to do is call the `grob()` function and supply all of the information that we want to record about `"box"` objects. In this case, we just record the labels to be drawn within the box and the location where we want to draw the box.

This function does not draw anything. For example, the following code creates two `"box"` objects, but produces no graphical output whatsoever.

```
> box1 <- boxGrob(c("ISBN", "title",
            "pub"), x=0.3)
> box2 <- boxGrob(c("ID", "name",
            "country"), x=0.7)
```

The `grid.draw()` function can be used to draw any graphical object, but we need to supply the details of how `"box"` objects get drawn. This is the purpose of the second function in Figure 2. This function is a *method* for the `drawDetails()` function; it says how to draw `"box"` objects. In this case, the function is very simple because it can call the `tableBox()` function that we defined in Figure 1. The important detail is that the `boxGrob()` function specified a special *class*, `cl="box"`, for `"box"` objects, which meant that we could define a `drawDetails()` method specifically for this sort of object and control what gets drawn.

With this `drawDetails()` method defined, we can draw the boxes that we created earlier by calling the `grid.draw()` function. This function will draw any **grid** graphical object by calling the appropriate method for the `drawDetails()` generic function (among other things). The following code calls `grid.draw()` to draw the two boxes.

```
> grid.draw(box1)
> grid.draw(box2)
```

At this point, we appear to have achieved only a more complicated equivalent of the previous graphics function. However, there are a number of other functions that can do useful things with **grid** graphical objects. For example, the `grobX()` and `grobY()` functions can be used to calculate locations on the boundary of a graphical object. As with `grid.draw()`, which has to call `drawDetails()` to find out how to draw a particular class of graphical object, these functions call generic functions to find out how to calculate locations on the boundary for a particular class of object. The generic functions are called `xDetails()` and `yDetails()` and methods for our special `"box"` class are defined in the last two functions in Figure 2.

These methods work by passing the buck. They both create a rounded rectangle at the correct location and the right size for the box, then call `grobX()` (or `grobY()`) to determine a location on the boundary of the rounded rectangle. In other words, they rely on code within the **grid** package that already exists to calculate the boundary of rounded rectangles.

With these methods defined, we are now in a position to draw a curved line between our boxes. The key idea is that we can use `grobX()` and `grobY()` to specify a start and end point for the curve. For example, we can start the curve at the right hand edge of

```
> tableBox <- function(labels, x=.5, y=.5) {
      nlabel <- length(labels)
      tablevp <-
          viewport(x=x, y=y,
                   width=max(stringWidth(labels)) +
                         unit(4, "mm"),
                   height=unit(nlabel, "lines"))
      pushViewport(tablevp)
      grid.roundrect()
      if (nlabel > 1) {
          for (i in 1:(nlabel - 1)) {
              fill <- c("white", "grey")[i %% 2 + 1]
              grid.clip(y=unit(i, "lines"), just="bottom")
              grid.roundrect(gp=gpar(fill=fill))
          }
      }
      grid.clip()
      grid.text(labels,
                x=unit(2, "mm"), y=unit(nlabel:1 - .5, "lines"),
                just="left")
      popViewport()
  }
```

Figure 1: A function to draw a diagram box, for a given set of `labels`, centred at the specified (`x`, `y`) location.

```
> boxGrob <- function(labels, x=.5, y=.5) {
      grob(labels=labels, x=x, y=y, cl="box")
  }
> drawDetails.box <- function(x, ...) {
      tableBox(x$labels, x$x, x$y)
  }
> xDetails.box <- function(x, theta) {
      nlines <- length(x$labels)
      height <- unit(nlines, "lines")
      width <- unit(4, "mm") + max(stringWidth(x$labels))
      grobX(roundrectGrob(x=x$x, y=x$y, width=width, height=height),
            theta)
  }
> yDetails.box <- function(x, theta) {
      nlines <- length(x$labels)
      height <- unit(nlines, "lines")
      width <- unit(4, "mm") + max(stringWidth(x$labels))
      grobY(rectGrob(x=x$x, y=x$y, width=width, height=height),
            theta)
  }
```

Figure 2: Some functions that define a graphical object representing a diagram box. The `boxGrob()` function constructs a `"box"` object, the `drawDetails()` method describes how to draw a `"box"` object, and the `xDetails()` and `yDetails()` functions calculate locations on the boundary of a `"box"` object.

box1 by specifying grobX(box1, "east"). The vertical position is slightly trickier because we do not want the line starting at the top or bottom of the box, but we can simply add or subtract the appropriate number of lines of text to get the right spot.

The following code uses these ideas to draw a curve from the pub label of box1 to the ID label of box2. The curve has two corners (inflect=TRUE) and it has a small arrow at the end.

This call to grid.curve() is relatively verbose, but in a diagram containing many similar curves, this burden can be significantly reduced by writing a simple function that hides away the common features, such as the specification of the arrow head.

The major gain from this object-oriented approach is that the start and end points of this curve are described by simple expressions that will automatically update if the locations of the boxes are modified.

```
> grid.curve(grobX(box1, "east"),
             grobY(box1, "south") +
               unit(0.5, "lines"),
             grobX(box2, "west"),
             grobY(box2, "north") -
               unit(0.5, "lines"),
             inflect=TRUE,
             arrow=
               arrow(type="closed",
                  angle=15,
                  length=unit(2, "mm")),
             gp=gpar(fill="black"))
```

Conclusion

This article has demonstrated a number of useful *low-level* graphical facilities in R with an example of how they can be combined to produce a diagram consisting of non-trivial nodes with smooth curves between them.

The code examples provided in this article have ignored some details in order to keep things simple. For example, there are no checks that the arguments have sensible values in the functions tableBox() and boxGrob(). However, for creating one-off diagrams, this level of detail is not necessary anyway.

One detail that would be encountered quite quickly in practice, in this particular sort of diagram, is that a curve from one box to another that needs to go across-and-down rather than across-and-up would require the addition of curvature=-1 to the grid.curve() call. Another thing that is missing is complete code to produce the example diagram from the beginning of this article, where there are five interconnected boxes and the boxes have some additional features, such as a distinct "header" line at the top. This complete code was excluded to save on space, but a simple R package is provided at http://www.stat.auckland.ac.nz/~paul/ with code to draw that complete diagram and the package also contains a more complete implementation of code to create and draw "box" graphical objects.

One final point is that using R graphics to draw diagrams like this is *not* fast. In keeping with the S tradition, the emphasis is on developing *code* quickly and on having code that is not a complete nightmare to maintain. In this case particularly, the speed of developing a diagram comes at the expense of the time taken to draw the diagram. For small, one-off diagrams this is not likely to be an issue, but the approach described in this article would *not* be appropriate, for example, for drawing a node-and-edge graph of the Internet.

Bibliography

C. Blanc and C. Schlick. X-splines: a spline model designed for the end-user. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 377–386, New York, NY, USA, 1995. ACM. ISBN 0-89791-701-4. doi: http://doi.acm.org/10.1145/218380.218488.

G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006. URL http://igraph.sf.net.

J. Gentry, L. Long, R. Gentleman, S. Falcon, F. Hahne, and D. Sarkar. *Rgraphviz: Provides plotting capabilities for R graph objects*, 2008. R package version 1.18.1.

A. Larsson. Dia, 2008. http://www.gnome.org/projects/dia/.

P. Murrell. The grid graphics package. *R News*, 2(2): 14–19, June 2002. URL http://CRAN.R-project.org/doc/Rnews/.

P. Murrell. Recent changes in grid graphics. *R News*, 5(1):12–20, May 2005a. URL http://CRAN.R-project.org/doc/Rnews/.

P. Murrell. *R Graphics*. Chapman & Hall/CRC, 2005b. URL http://www.stat.auckland.ac.nz/~paul/RGraphics/rgraphics.html. ISBN 1-584-88486-X.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. URL http://www.R-project.org. ISBN 3-900051-07-0.

D. Sarkar. *Lattice: Multivariate Data Visualization with R*. Springer, 2008. URL http://amazon.com/o/ASIN/0387759689/. ISBN 9780387759685.

K. Soetaert. *diagram: Functions for visualising simple graphs (networks), plotting flow diagrams*, 2008a. R package version 1.2.

K. Soetaert. *shape: Functions for plotting graphical shapes, colors*, 2008b. R package version 1.2.

H. Wickham. *ggplot2*. Springer, 2009. http://had.co.nz/ggplot2/.

Paul Murrell
Department of Statistics
The University of Auckland
New Zealand
paul@stat.auckland.ac.nz

The hwriter Package

Composing HTML documents with R objects

by Gregoire Pau and Wolfgang Huber

Introduction

HTML documents are structured documents made of diverse elements such as paragraphs, sections, columns, figures and tables organized in a hierarchical layout. Combination of HTML documents and hyperlinking is useful to report analysis results; for example, in the package **arrayQualityMetrics** (Kauffmann et al., 2009), estimating the quality of microarray data sets and **cellHTS2** (Boutros et al., 2006), performing the analysis of cell-based screens.

There are several tools for exporting data from R into HTML documents. The package **R2HTML** is able to render a large diversity of R objects in HTML but does not easily support combining them in a structured layout and has a complex syntax. On the other hand, the package **xtable** can render R matrices with simple commands but cannot combine HTML elements and has limited formatting options.

The package **hwriter** allows rendering R objects in HTML and combining resulting elements in a structured layout. It uses a simple syntax, supports extensive formatting options and takes full advantage of the ellipsis `'...'` argument and R vector recycling rules.

Comprehensive documentation and examples of **hwriter** are generated by running the command `example(hwriter)`, which creates the package web page `http://www.ebi.ac.uk/~gpau/hwriter`.

The function `hwrite`

The generic function `hwrite` is the core function of the package **hwriter**. `hwrite(x, page=NULL, ...)` renders the object `x` in HTML using the formatting arguments specified in `'...'` and returns a character vector containing the HTML element code of `x`.

If `page` is a filename, the HTML element is written in this file. If it is an R connection/file, the element is appended to this connection, allowing the sequential building of HTML documents. If `NULL`, the returned HTML element code can be concatenated with other elements with `paste` or nested inside other elements through subsequent `hwrite` calls. These operations are the tree structure equivalents of adding a sibling node and adding a child node in the document tree.

Formatting objects

The most basic call of `hwrite` is to output a character vector into an HTML document.

```
> hwrite('Hello world !', 'doc.html')
```

Hello world !

Character strings can be rendered with a variety of formatting options. The argument `link` adds a hyperlink to the HTML element pointing to an external document. The argument `style` specifies an inline Cascaded Style Sheet (CSS) style (color, alignment, margins, font, ...) to render the element. Other arguments allow a finer rendering control.

```
> hwrite('Hello world !', 'doc.html',
  link='http://cran.r-project.org/')
```

Hello world !

```
> hwrite('Hello world !', 'doc.html',
  style='color: red; font-size: 20pt;
  font-family: Gill Sans Ultra Bold')
```

Hello world !

Images can be included using `hwriteImage` which supports diverse formatting options.

```
> img=system.file('images', c('iris1.jpg',
  'iris3.jpg'), package='hwriter')
> hwriteImage(img[2], 'doc.html')
```

Arguments recycling

Objects written with formatting arguments containing more than one element are recycled by `hwrite`, producing a vector of HTML elements.

```
> hwrite('test', 'doc.html', style=paste(
  'color:', c('peru', 'purple', 'green')))
```

test test test

Vectors and matrices

Vectors and matrices are rendered as HTML tables.

```
> hwrite(iris[1:2, 1:2], 'doc.html')
```

	Sepal.Length	Sepal.Width
1	5.1	3.5
2	4.9	3

Formatting arguments are recycled and distributed over table cells. Arguments `link` and `style` are valid. Cell background color is controlled by the argument `bgcolor`.

```
> colors=rgb(colorRamp(c('red', 'yellow',
  'white'))((0:7)/7), max=255)
> hwrite(0:7, 'doc.html', bgcolor=colors,
  style='padding:5px')
```

0	1	2	3	4	5	6	7

Formatting arguments can also be distributed on rows and columns using the syntax `row.*` and `col.*` where `*` can be any valid HTML <td> attribute. Global table properties are specified using the syntax `table.*` where `*` can be any valid HTML <table> attribute. As an example, `table.style` controls the global table CSS inline style and `table.cellpadding` controls the global cell padding size.

```
> hwrite(iris[1:2, 1:2], 'doc.html',
  row.bgcolor='#ffdc98', table.style=
  'border-collapse:collapse',
  table.cellpadding='5px')
```

	Sepal.Length	Sepal.Width
1	5.1	3.5
2	4.9	3

Appending HTML elements to a page

A new HTML document is created by the function `openPage` which returns an R connection. Appending HTML elements to the page is done by passing the connection to `hwrite` instead of a filename. The document is closed by `closePage` which terminates the HTML document. The function `openPage` supports the argument `link.css` which links the document to external stylesheets containing predefined CSS styles and classes. Classes can be used with the argument `class` of the function `hwrite`.

```
> hcss=system.file('images', 'hwriter.css',
  package='hwriter')
> p=openPage('doc.html', link.css=hcss)
> hwriteImage(img, p, br=TRUE)
> hwrite(iris[1:2, 1:2], p, br=TRUE,
  row.bgcolor='#d2c0ed')
> hwrite('Iris flowers', p, class='intro')
> closePage(p)
```


	Sepal.Length	Sepal.Width
1	5.1	3.5
2	4.9	3

Iris flowers

Nesting HTML elements

The function `hwrite` always returns a character vector containing HTML code parts. Code parts can be reused in further `hwrite` calls to build tables containing complex cells or inner tables, which are useful to compose HTML document layouts.

```
> cap=hwrite(c('Plantae','Monocots','Iris'),
  table.style='border-collapse:collapse',
  table.cellpadding='5px')
> print(cap)

<table style="border-collapse:collapse"
cellpadding="5px" border="1"><tr><td>
Plantae</td><td>Monocots</td><td>Iris
</td></tr></table>

> hwrite(matrix(c(hwriteImage(img[2]),cap)),
  'doc.html', table.cellpadding='5px',
  table.style='border-collapse:collapse')
```

Plantae	Monocots	Iris

Bibliography

M. Boutros, L. Bras, and W. Huber. Analysis of cell-based RNAi screens. *Genome Biology*, 7(66), 2006.

A. Kauffmann, R. Gentleman, and W. Huber. arrayQualityMetrics - a Bioconductor package for quality assessment of microarray data. *Bioinformatics*, 25(3), 2009.

Gregoire Pau
EMBL-European Bioinformatics Institute
Cambridge, UK
gregoire.pau@ebi.ac.uk

Wolfgang Huber
EMBL-European Bioinformatics Institute
Cambridge, UK
huber@ebi.ac.uk

AdMit: Adaptive Mixtures of Student-t Distributions

by David Ardia, Lennart F. Hoogerheide and Herman K. van Dijk

Introduction

This note presents the package **AdMit** (Ardia et al., 2008, 2009), an R implementation of the adaptive mixture of Student-t distributions (AdMit) procedure developed by Hoogerheide (2006); see also Hoogerheide et al. (2007); Hoogerheide and van Dijk (2008). The AdMit strategy consists of the construction of a mixture of Student-t distributions which approximates a target distribution of interest. The fitting procedure relies only on a kernel of the target density, so that the normalizing constant is not required. In a second step, this approximation is used as an importance function in importance sampling or as a candidate density in the independence chain Metropolis-Hastings (M-H) algorithm to estimate characteristics of the target density. The estimation procedure is fully automatic and thus avoids the difficult task, especially for non-experts, of tuning a sampling algorithm. Typically, the target is a posterior distribution in a Bayesian analysis, where we indeed often only know a kernel of the posterior density.

In a *standard* case of importance sampling or the independence chain M-H algorithm, the candidate density is unimodal. If the target distribution is multimodal then some draws may have huge weights in the importance sampling approach and a second mode may be completely missed in the M-H strategy. As a consequence, the convergence behavior of these Monte Carlo integration methods is rather uncertain. Thus, an important problem is the choice of the importance or candidate density, especially when little is known a priori about the shape of the target density. For both importance sampling and the independence chain M-H, it holds that the candidate density should be *close* to the target density, and it is especially important that the tails of the candidate should not be thinner than those of the target.

Hoogerheide (2006) and Hoogerheide et al. (2007) mention several reasons why mixtures of Student-t distributions are natural candidate densities. First, they can provide an accurate approximation to a wide variety of target densities, with substantial skewness and high kurtosis. Furthermore, they can deal with multi-modality and with non-elliptical shapes due to asymptotes. Second, this approximation can be constructed in a quick, iterative procedure and a mixture of Student-t distributions is easy to sample from. Third, the Student-t distribution has fatter tails than the Normal distribution; especially if one specifies Student-t distributions with few degrees of freedom, the risk is small that the tails of the candidate are thinner than those of the target distribution. Finally, Zeevi and Meir (1997) showed that under certain conditions any density function may be approximated to arbitrary accuracy by a convex combination of *basis* densities; the mixture of Student-t distributions falls within their framework.

The package **AdMit** consists of three main functions: `AdMit`, `AdMitIS` and `AdMitMH`. The first one allows the user to fit a mixture of Student-t distributions to a given density through its kernel function. The next two functions perform importance sampling and independence chain M-H sampling using the fitted mixture estimated by `AdMit` as the importance or candidate density, respectively.

To illustrate the use of the package, we apply the AdMit methodology to a bivariate bimodal distribution. We describe the use of the functions provided by the package and document the ability and relevance of the methodology to reproduce the shape of non-elliptical distributions.

Illustration

This section presents the functions provided by the package **AdMit** with an illustration of a bivariate bimodal distribution. This distribution belongs to the class of conditionally Normal distributions proposed by Gelman and Meng (1991) with the property that the joint density is not Normal. It is not a posterior distribution, but it is chosen because it is a simple distribution with non-elliptical shapes so that it allows for an easy illustration of the AdMit approach.

Let X_1 and X_2 be two random variables, for which X_1 is Normally distributed given X_2 and vice versa. Then, the joint distribution, after location and scale transformations in each variable, can be written as (see Gelman and Meng, 1991):

$$p(\mathbf{x}) \propto k(\mathbf{x}) \doteq \exp\Big[-\tfrac{1}{2}\big(Ax_1^2x_2^2 + x_1^2 + x_2^2 - 2Bx_1x_2 - 2C_1x_1 - 2C_2x_2\big)\Big] \tag{1}$$

where $p(\mathbf{x})$ denotes a density, $k(\mathbf{x})$ a kernel and $\mathbf{x} \doteq (x_1, x_2)'$ for notational purposes. A, B, C_1 and C_2 are constants; we consider an asymmetric case in which $A = 5$, $B = 5$, $C_1 = 3$, $C_2 = 3.5$ in what follows.

The adaptive mixture approach determines the number of mixture components H, the mixing probabilities, the modes and scale matrices of the components in such a way that the mixture density $q(\mathbf{x})$ approximates the target density $p(\mathbf{x})$ of which we only

know a kernel function $k(\mathbf{x})$ with $\mathbf{x} \in \mathbb{R}^d$. Typically, $k(\mathbf{x})$ will be a posterior density kernel for a vector of model parameters $\mathbf{x}$.

The AdMit strategy consists of the following steps:

(0) Initialization: computation of the mode and scale matrix of the first component, and drawing a sample from this Student-t distribution;

(1) Iterate on the number of components: add a new component that covers a part of the space of $\mathbf{x}$ where the previous mixture density was relatively small, as compared to $k(\mathbf{x})$;

(2) Optimization of the mixing probabilities;

(3) Drawing a sample $\{\mathbf{x}^{[i]}\}$ from the new mixture $q(\mathbf{x})$;

(4) Evaluation of importance sampling weights $\{k(\mathbf{x}^{[i]})/q(\mathbf{x}^{[i]})\}$. If the coefficient of variation, i.e. the standard deviation divided by the mean, of the weights has converged, then stop. Otherwise, go to step (1). By default, the adaptive procedure stops if the relative change in the coefficient of variation of the importance sampling weights caused by adding one new Student-t component to the candidate mixture is less than 10%.

There are two main reasons for using the coefficient of variation of the importance sampling weights as a measure of the fitting quality. First, it is a natural, intuitive measure of quality of the candidate as an approximation to the target. If the candidate and the target distributions coincide, all importance sampling weights are equal, so that the coefficient of variation is zero. For a poor candidate that not even roughly approximates the target, some importance sampling weights are huge while most are (almost) zero, so that the coefficient of variation is high. The better the candidate approximates the target, the more evenly the weight is divided among the candidate draws, and the smaller the coefficient of variation of the importance sampling weights. Second, Geweke (1989) argues that a reasonable objective in the choice of an importance density is the minimization of the variance, or equivalently the coefficient of variation, of the importance weights. We prefer to quote the coefficient of variation because it does not depend on the number of draws or the integration constant of the posterior density kernel, unlike the standard deviation of the scaled or unscaled importance weights, respectively. The coefficient of variation merely reflects the quality of the candidate as an approximation to the target.

Note also that all Student-t components in the mixture approximation $q(\mathbf{x})$ have the same degrees of freedom parameter. Moreover, this value is not determined by the procedure but set by the user; by default, the algorithm uses the value one (i.e. the approximation density is a mixture of Cauchy) since i) it enables the method to deal with fat-tailed target distributions and ii) it makes it easier for the iterative procedure to detect modes that are far apart. There may be a trade-off between efficiency and robustness at this point: setting the degrees of freedom to one reflects that we find a higher robustness more valuable than a slight possible gain in efficiency.

The core function provided by the package is the function `AdMit`. The main arguments of the function are: `KERNEL`, a kernel function of the target density on which the approximation is constructed. This function must contain the logical argument `log`. When `log = TRUE`, the function `KERNEL` returns the logarithm value of the kernel function; this is used for numerical stability. `mu0` is the starting value of the first stage optimization; it is a vector whose length corresponds to the length of the first argument in `KERNEL`. `Sigma0` is the (symmetric positive definite) scale matrix of the first component. If a matrix is provided by the user, it is used as the scale matrix of the first component and then `mu0` is used as the mode of the first component (instead of a starting value for optimization). `control` is a list of tuning parameters, containing in particular: `df` (default: 1), the degrees of freedom of the mixture components and `CVtol` (default: 0.1), the tolerance of the relative change of the coefficient of variation. For further details, the reader is referred to the documentation manual (by typing `?AdMit`) or to Ardia et al. (2009). Finally, the last argument of `AdMit` is `...` which allows the user to pass additional arguments to the function `KERNEL`.

Let us come back to our bivariate conditionally Normal distribution. First, we need to code the kernel:

```
> GelmanMeng <- function(x, log = TRUE)
+ {
+   if (is.vector(x))
+     x <- matrix(x, nrow = 1)
+   r <- -0.5 * ( 5 * x[,1]^2 * x[,2]^2
+                 + x[,1]^2 + x[,2]^2
+                 - 10 * x[,1] * x[,2]
+                 - 6 * x[,1] - 7 * x[,2] )
+   if (!log)
+     r <- exp(r)
+   as.vector(r)
+ }
```

Note that the argument `log` is set to `TRUE` by default so that the function outputs the logarithm of the kernel. Moreover, the function is vectorized to speed up the computations. The contour plot of `GelmanMeng` is displayed in the left part of Figure 1.

We now use the function `AdMit` to find a suitable approximation for the density whose kernel is `GelmanMeng`. We use the starting value `mu0 = c(0, 0.1)` for the first stage optimization and assign the result of the function to `outAdMit`.

```
> set.seed(1234)
```

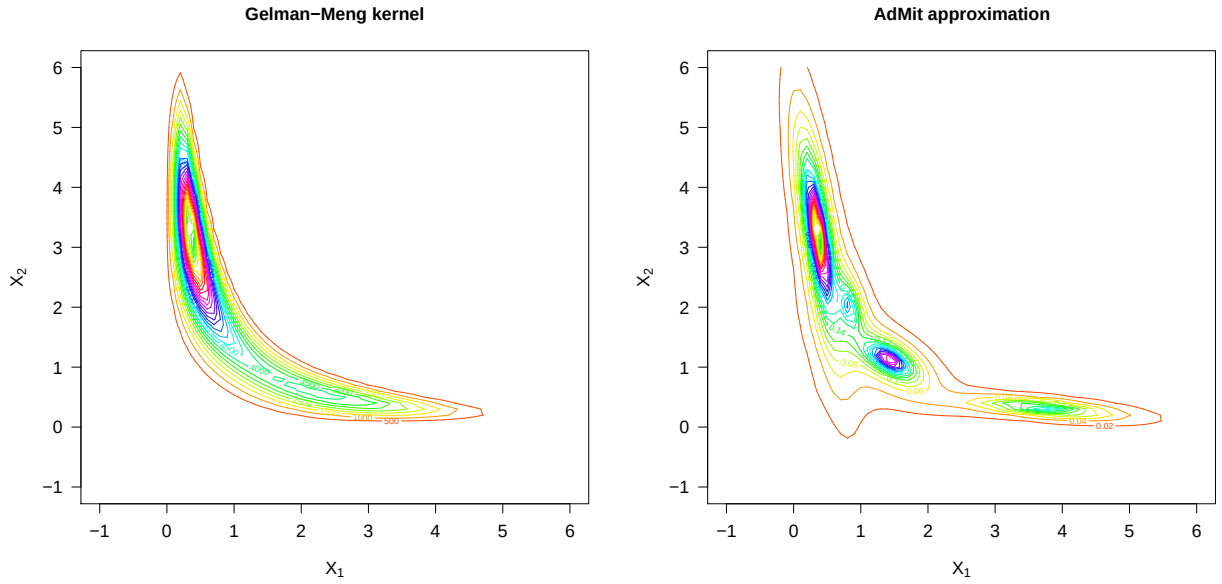


Figure 1: Left: contour plot of the kernel `GelmanMeng`. Right: contour plot of the four-component Student-t mixture approximation estimated by the function `AdMit`.

```
> outAdMit <- AdMit(KERNEL = GelmanMeng,
+                   mu0 = c(0, 0.1))

$CV
[1] 8.0737 1.7480 0.9557 0.8807

$mit
$mit$p
   cmp1    cmp2    cmp3    cmp4
0.54707 0.14268 0.21053 0.09972

$mit$mu
          k1     k2
cmp1 0.3640 3.2002
cmp2 3.7088 0.3175
cmp3 1.4304 1.1240
cmp4 0.8253 2.0261

$mit$Sigma
        k1k1     k1k2     k2k1    k2k2
cmp1 0.03903 -0.15606 -0.15606 1.22561
cmp2 0.86080 -0.08398 -0.08398 0.02252
cmp3 0.13525 -0.06743 -0.06743 0.09520
cmp4 0.03532 -0.04072 -0.04072 0.21735

$mit$df
[1] 1

$summary
  H METHOD.mu TIME.mu METHOD.p TIME.p     CV
1 1      BFGS    0.21      NONE   0.00 8.0737
2 2      BFGS    0.08    NLMINB   0.08 1.7480
3 3      BFGS    0.12    NLMINB   0.08 0.9557
4 4      BFGS    0.20    NLMINB   0.19 0.8807
```

The output of the function `AdMit` is a list. The first component is `CV`, a vector of length H which gives the value of the coefficient of variation of the importance sampling weights at each step of the adaptive fitting procedure. The second component is `mit`, a list which consists of four components giving information on the fitted mixture of Student-t distributions: `p` is a vector of length H of mixing probabilities, `mu` is a $H \times d$ matrix whose rows give the modes of the mixture components, `Sigma` is a $H \times d^2$ matrix whose rows give the scale matrices (in vector form) of the mixture components and `df` is the degrees of freedom of the Student-t components. The third component of the list returned by `AdMit` is `summary`. This is a data frame containing information on the adaptive fitting procedure. We refer the reader to the documentation manual for further details.

For the kernel `GelmanMeng`, the approximation constructs a mixture of four components. The value of the coefficient of variation decreases from 8.0737 to 0.8807. A contour plot of the four-component approximation is displayed in the right-hand side of Figure 1. This graph is produced using the function `dMit` which returns the density of the mixture given by the output `outAdMit$mit`. The contour plot illustrates that the four-component mixture provides a good approximation of the density: the areas with substantial target density mass are *covered* by sufficient density mass of the mixture approximation.

Once the adaptive mixture of Student-t distributions is fitted to the density using a kernel, the approximation provided by `AdMit` is used as the importance sampling density in importance sampling or as the candidate density in the independence chain M-H algorithm.

The **AdMit** package contains the `AdMitIS` function which performs importance sampling using the mixture approximation in `outAdMit$mit` as the im-

portance density. The arguments of the function AdMitIS are: (i) N, the number of draws used in importance sampling; (ii) KERNEL, a kernel function of the target density; (iii) G, the function of which the expectation $\mathbb{E}_p\left[g(\mathbf{x})\right]$ is estimated; By default, the function G is the identity so that the function outputs a vector containing the mean estimates for the components of $\mathbf{x}$. Alternative functions may be provided by the user to obtain other quantities of interest for $p(\mathbf{x})$. The only requirement is that the function outputs a matrix; (iv) mit, a list providing information on the mixture approximation; (v) ... allows additional parameters to be passed to the function KERNEL and/or G.

Let us apply the function AdMitIS to the kernel GelmanMeng using the approximation outAdMit$mit:

```
> outAdMitIS <- AdMitIS(N = 1e5,
+                       KERNEL = GelmanMeng,
+                       mit = outAdMit$mit)

$ghat
[1] 0.9556 2.2465

$NSE
[1] 0.003833 0.005639

$RNE
[1] 0.6038 0.5536
```

The output of the function AdMitIS is a list. The first component is ghat, the importance sampling estimator of $\mathbb{E}_p\left[g(\mathbf{x})\right]$. The second component is NSE, a vector containing the numerical standard errors (the variation of the estimates that can be expected if the simulations were to be repeated) of the components of ghat. The third component is RNE, a vector containing the relative numerical efficiencies of the components of ghat (the ratio between an estimate of the variance of an estimator based on direct sampling and the importance sampling estimator's estimated variance with the same number of draws). RNE is an indicator of the efficiency of the chosen importance function; if target and importance densities coincide, RNE equals one, whereas a very poor importance density will have a RNE close to zero. Both NSE and RNE are estimated by the method given in Geweke (1989).

Further, the **AdMit** package contains the AdMitMH function which uses the mixture approximation in outAdMit$mit as the candidate density in the independence chain M-H algorithm. The arguments of the function AdMitMH are: (i) N, the length of the MCMC sequence of draws; (ii) KERNEL, a kernel function of the target density; (iii) mit, a list providing information on the mixture approximation; (iv) ... allows additional parameters to be passed to the function KERNEL.

Let us apply the function AdMitMH to the kernel GelmanMeng using the approximation outAdMit$mit:

```
> outAdMitMH <- AdMitMH(N = 101000,
+                       KERNEL = GelmanMeng,
+                       mit = outAdMit$mit)

$draws
               k1          k2
1       7.429e-01  3.021e-01
2       7.429e-01  3.021e-01
3       7.429e-01  3.021e-01
4       1.352e+00  1.316e+00
5       1.011e+00  1.709e+00
6       1.011e+00  1.709e+00
7       1.005e+00  1.386e+00
...

$accept
[1] 0.5119
```

The output of the function AdMitMH is a list of two components. The first component is draws, a $N \times d$ matrix containing draws from the target density $p(\mathbf{x})$ in its rows. The second component is accept, the acceptance rate of the independence chain M-H algorithm.

The package **coda** (Plummer et al., 2008) can be used to check the convergence of the MCMC chain and obtain quantities of interest for $p(\mathbf{x})$. Here, for simplicity, we discard the first 1'000 draws as a burn-in sample and transform the output outAdMitMH$draws in a mcmc object using the function as.mcmc provided by **coda**. A summary of the MCMC chain can be obtained using summary:

```
> library("coda")
> draws <- as.mcmc(outAdMitMH$draws)
> draws <- window(draws, start = 1001)
> colnames(draws) <- c("X1", "X2")
> summary(draws)$stat

     Mean     SD Naive SE Time-series SE
X1 0.9643 0.9476 0.002996       0.004327
X2 2.2388 1.3284 0.004201       0.006232
```

We note that the mean estimates are close to the values obtained with the function AdMitIS. The relative numerical efficiency (RNE) can be computed using the functions effectiveSize and niter:

```
> effectiveSize(draws) / niter(draws)

    X1     X2
0.3082 0.3055
```

These relative numerical efficiencies reflect the good quality of the candidate density in the independence chain M-H algorithm.

The approach is compared to the standard Gibbs sampler, which is extremely easy to implement here since the full conditional densities are Normals. We generated $N = 101000$ draws from a Gibbs sampler using the first 1'000 draws as a burn-in period. In this case, the RNE were 0.06411 and 0.07132, respectively. In Figure 2 we display the autocorrelogram for the MCMC output of the AdMitMH function (upper graphs) together with the autocorrelogram of the Gibbs (lower part). We clearly notice that the Gibbs sequence is mixing much more slowly, explaining the

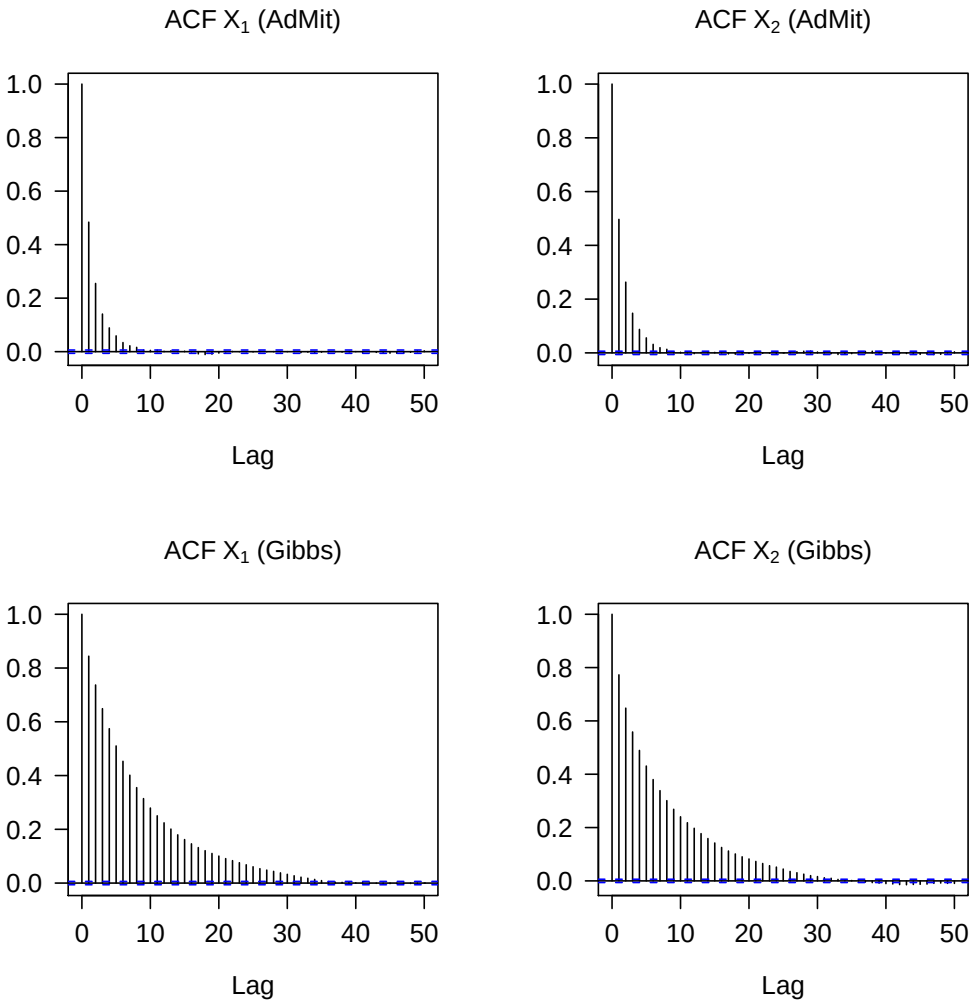


Figure 2: Upper graphs: Autocorrelation function (ACF) for the `AdMitMH` output. Lower graphs: ACF for the Gibbs output.

lower RNE values. This example illustrates the phenomenon that the autocorrelation in the Gibbs sequence may be high when assessing a non-elliptical distribution, and that the AdMit approach may overcome this.

Summary

This note presented the package **AdMit** which provides functions to approximate and sample from a certain target distribution given only a kernel of the target density function. The estimation procedure is fully automatic and thus avoids the time-consuming and difficult task of tuning a sampling algorithm. The use of the package has been illustrated in a bivariate bimodal distribution.

Interested users are referred to Ardia et al. (2009) for a more complete discussion on the **AdMit** package. The article provides a more extensive introduction of the package usage as well as an illustration of the relevance of the AdMit procedure with the Bayesian estimation of a mixture of ARCH model fitted to foreign exchange log-returns data. The methodology is compared to standard cases of importance sampling and the Metropolis-Hastings algorithm using a naive candidate and with the Griddy-Gibbs approach. It is shown that for investigating means and particularly tails of the joint posterior distribution the AdMit approach is preferable.

Acknowledgments

The authors acknowledge an anonymous reviewer for helpful comments that have led to improvements of this note. The first author is grateful to the Swiss National Science Foundation (under grant #FN PB FR1-121441) for financial support. The third author gratefully acknowledges the financial assistance from the Netherlands Organization of Research (under grant #400-07-703).

Bibliography

D. Ardia, L. F. Hoogerheide, and H. K. van Dijk. **AdMit**: *Adaptive Mixture of Student-t Distributions for Efficient Simulation in R*, 2008. URL http://CRAN.R-project.org/package=AdMit. R package version 1.01-01.

D. Ardia, L. F. Hoogerheide, and H. K. van Dijk. Adaptive mixture of Student-t distributions as a flexible candidate distribution for efficient simulation: The R package **AdMit**. *Journal of Statistical Software*, 29(3):1–32, 2009. URL http://www.jstatsoft.org/v29/i03/.

A. Gelman and X.-L. Meng. A note on bivariate distributions that are conditionally normal. *The American Statistician*, 45(2):125–126, 1991.

J. F. Geweke. Bayesian inference in econometric models using Monte Carlo integration. *Econometrica*, 57 (6):1317–1339, 1989.

L. F. Hoogerheide. *Essays on Neural Network Sampling Methods and Instrumental Variables*. PhD thesis, Tinbergen Institute, Erasmus University Rotterdam, 2006. Book nr. 379 of the Tinbergen Institute Research Series.

L. F. Hoogerheide and H. K. van Dijk. Possibly ill-behaved posteriors in econometric models: On the connection between model structures, non-elliptical credible sets and neural network simulation techniques. Technical Report 2008-036/4, Tinbergen Institute, Erasmus University Rotterdam, 2008. URL http://www.tinbergen.nl/discussionpapers/08036.pdf.

L. F. Hoogerheide, J. F. Kaashoek, and H. K. van Dijk. On the shape of posterior densities and credible sets in instrumental variable regression models with reduced rank: An application of flexible sampling methods using neural networks. *Journal of Econometrics*, 139(1):154–180, 2007.

M. Plummer, N. Best, K. Cowles, and K. Vines. **coda**: *Output Analysis and Diagnostics for MCMC in R*, 2008. URL http://CRAN.R-project.org/package=coda. R package version 0.13-3.

A. J. Zeevi and R. Meir. Density estimation through convex combinations of densities: Approximation and estimation bounds. *Neural Networks*, 10(1):99–109, 1997.

David Ardia
University of Fribourg, Switzerland
david.ardia@unifr.ch

Lennart F. Hoogerheide
Erasmus University Rotterdam, The Netherlands

Herman K. van Dijk
Erasmus University Rotterdam, The Netherlands

expert: Modeling Without Data Using Expert Opinion

by Vincent Goulet, Michel Jacques and Mathieu Pigeon

Introduction

The **expert** package provides tools to create and manipulate empirical statistical models using expert opinion (or judgment). Here, the latter expression refers to a specific body of techniques to elicit the distribution of a random variable when data is scarce or unavailable. Opinions on the quantiles of the distribution are sought from experts in the field and aggregated into a final estimate. The package supports aggregation by means of the Cooke, Mendel–Sheridan and predefined weights models.

We do not mean to give a complete introduction to the theory and practice of expert opinion elicitation in this paper. However, for the sake of completeness and to assist the casual reader, the next section summarizes the main ideas and concepts.

It should be noted that we are only interested, here, in the mathematical techniques of expert opinion aggregation. Obtaining the opinion from the experts is an entirely different task; see Kadane and Wolfson (1998); Kadane and Winkler (1988); Tversky and Kahneman (1974) for more information. Moreover, we do not discuss behavioral models (see Ouchi, 2004, for an exhaustive review) nor the problems of expert selection, design and conducting of interviews. We refer the interested reader to O'Hagan et al. (2006) and Cooke (1991) for details. Although it is extremely important to carefully examine these considerations if expert opinion is to be useful, we assume that these questions have been solved previously. The package takes the opinion of experts as an input that we take here as available.

The other main section presents the features of version 1.0-0 of package **expert**.

Expert opinion

Consider the task of modeling the distribution of a quantity for which data is very scarce (e.g. the cost of a nuclear accident) or even unavailable (e.g. settlements in liability insurance, usually reached out of court and kept secret). A natural option for the analyst[1] is then to ask experts their opinion on the distribution of the so-called *decision variable*. The experts are people who have a good knowledge of the field under study and who are able to express their opinion in a simple probabilistic fashion. They can be engineers, physicists, lawyers, actuaries, etc. Typically, a group of 12 to 15 experts is consulted.

The experts express their opinion on the distribution of the decision random variable by means of a few quantiles, usually the median and a symmetric interval around it. Let us recall that the $100rth$ quantile of a random variable is the value q_r such that

$$\lim_{h \to 0} F(q_r - h) \leq r \leq F(q_r),$$

where F is the cumulative distribution function of the random variable. Hence, by giving the quantiles q_u and q_v, $u < v$, an expert states that, in his opinion, there is a probability of $v - u$ that the true value of the variable lies between q_u and q_v. We generally avoid asking for quantiles far in the tails (say, below 5% and above 95%) since humans have difficulty evaluating them.

In addition to the distribution of the decision variable, the experts are also queried for the distribution of a series of *seed* (or calibration) variables for which only the analyst knows the true values. By comparing the distributions given by the experts to the latter, the analyst will be able to assess the quality of the information provided by each expert. This step is called *calibration*.

In summary, expert opinion is given by means of quantiles for k seed variables and one decision variable. The calibration phase determines the influence of each expert on the final *aggregated distribution*. The three methods discussed in this paper and supported by the package are simply different ways to aggregate the information provided by the experts.

The aggregated distribution in the classical model of Cooke (1991) is a convex combination of the quantiles given by the experts, with weights obtained from the calibration phase. Consequently, if we asked for three quantiles from the experts (say the $10th$, $50th$ and $90th$), the aggregated distribution will consist of three "average" quantiles corresponding to these same probabilities. This model has been used in many real life studies by the analysts of the Delft Institute of Applied Mathematics of the Delft University of Technology (Cooke and Goossens, 2008).

The predefined weights method is similar, except that the weights are provided by the analyst.

On the other hand, the model of Mendel and Sheridan (1989) adjusts the probabilities associated with each quantile by a (somewhat convoluted) Bayesian procedure to reflect the results of the calibration phase. Returning to the context above, this means that the probability of 10% associated with the first quantile given by an expert may be changed to, say, 80% if the calibration phase proved that this ex-

[1] Also called *decision maker* in the literature.

pert systematically overestimates the distributions. Combining intersecting adjusted probabilities from all the experts usually results in an aggregated distribution with far more quantiles than were asked for in the beginning.

It is well beyond the scope of this paper to discuss the actual assumptions and computations in the calibration and aggregation phases. The interested reader may consult the aforementioned references, Goulet et al. (2008) or Pigeon (2008, in French).

The expert package

To the best of our knowledge, the only readily available software for expert opinion calculations is Excalibur, a closed source, Windows-only application available from the Risk and Environmental Modelling website of the Delft University of Technology (http://dutiosc.twi.tudelft.nl/~risk/). Excalibur does not support the Mendel–Sheridan model. Data has to be entered manually in the application, making connection to other tools inconvenient. Furthermore, exporting the results for further analysis requires an ad hoc procedure.

expert is a small R package providing a unified interface to the three expert opinion models exposed above along with a few utility functions and methods to manipulate the results. Naturally, the analyst also benefits from R's rich array of statistical, graphical, import and export tools.

Example dataset

Before looking at the functions of the package, we introduce a dataset that will be used in forthcoming examples. We consider an analysis where three experts must express their opinion on the 10*th*, 50*th* and 90*th* quantiles of a decision variable and two seed variables. The true values of the seed variables are 0.27 and 210,000, respectively. See Table 1 for the quantiles given by the experts.

Main function

The main function of the package is `expert`. It serves as a unified interface to all three models supported by the package. The arguments of the function are the following:

1. `x`, the quantiles of the experts for all seed variables and the decision variable — in other words the content of Table 1. This is given as a list of lists, one for each expert. The interior lists each contain vectors of quantiles: one per seed variable and one for the decision variable (in this order). For the example above, this gives

```
> x <- list(
+   E1 <- list(
+     S1 <- c(0.14, 0.22, 0.28),
+     S2 <- c(130000, 150000, 200000),
+     X  <- c(350000, 400000, 525000)),
+   E2 <- list(
+     S1 <- c(0.2, 0.3, 0.4),
+     S2 <- c(165000, 205000, 250000),
+     X  <- c(550000, 600000, 650000)),
+   E3 <- list(
+     S1 <- c(0.2, 0.4, 0.52),
+     S2 <- c(200000, 400000, 500000),
+     X  <- c(625000, 700000, 800000)))
```

2. `method`, the model to be used for aggregation. Must be one of '`cooke`' for Cooke's classical model, '`ms`' for the Mendel–Sheridan model, or '`weights`' for the predefined weights model;

3. `probs`, the vector of probabilities corresponding to the quantiles given by the experts. In the example, we have

```
> probs <- c(0.1, 0.5, 0.9)
```

4. `true.seed`, the vector of true values of the seed variables. Obviously, these must be listed in the same order as the seed variables are listed in argument `x`. In the example, this is

```
> true.seed <- c(0.27, 210000)
```

5. `alpha`, parameter α in Cooke's model. This argument is ignored in the other models. Parameter α sets a threshold for the calibration component of an expert below which the expert automatically receives a weight of 0 in the aggregated distribution. If the argument is missing or `NULL`, the value of α is determined by an optimization procedure (see below);

6. `w`, the vector of weights in the predefined weights model. This argument is ignored in the other models. If the argument is missing or `NULL`, the weights are all equal to $1/n$, where n is the number of experts.

If the calibration threshold α in Cooke's model is not fixed by the analyst, one is computed following a procedure proposed by Cooke (1991). We first fit the model with $\alpha = 0$. This gives a weight to each expert. Using these weights, we then create a "virtual" expert by aggregating the quantiles for the seed variables. The optimal α is the value yielding the largest weight for the virtual expert. This is determined by trying all values of α just below the calibration components obtained in the first step. There is no need to try other values and, hence, to conduct a systematic numerical optimization.

When the experts do not provide the 0*th* and 100*th* quantiles, `expert` sets them automatically following the ad hoc procedure exposed in Cooke (1991). In a nutshell, the quantiles are obtained by

		Variable		
Expert	Quantile	Seed 1	Seed 2	Decision
1	10*th*	0.14	130,000	350,000
	50*th*	0.22	150,000	400,000
	90*th*	0.28	200,000	525,000
2	10*th*	0.20	165,000	550,000
	50*th*	0.30	205,000	600,000
	90*th*	0.40	250,000	650,000
3	10*th*	0.20	200,000	625,000
	50*th*	0.40	400,000	700,000
	90*th*	0.52	500,000	800,000

Table 1: Quantiles given by the experts for the seed and decision variables

removing and adding 10% of the smallest interval containing all quantiles given by the experts to the bounds of this interval. Therefore, the minimum and maximum of the distribution are set the same for all experts. In our example, the $0th$ and $100th$ quantiles of the first seed variable are set, respectively, to

$$q_0 = 0.14 - 0.1(0.52 - 0.14) = 0.102$$
$$q_1 = 0.52 + 0.1(0.52 - 0.14) = 0.558,$$

those of the second seed variable to

$$q_0 = 130{,}000 - 0.1(500{,}000 - 130{,}000) = 93{,}000$$
$$q_1 = 500{,}000 + 0.1(500{,}000 - 130{,}000) = 537{,}000$$

and, finally, those of the decision variable to

$$q_0 = 350{,}000 - 0.1(800{,}000 - 350{,}000) = 305{,}000$$
$$q_1 = 800{,}000 + 0.1(800{,}000 - 350{,}000) = 845{,}000.$$

Note that only the Mendel–Sheridan model allows non-finite lower and upper bounds.

The function `expert` returns a list of class '`expert`' containing the vector of the knots (or bounds of the intervals) of the aggregated distribution, the vector of corresponding probability masses and some other characteristics of the model used. A `print` method displays the results neatly.

Consider using Cooke's model with the data of the example and a threshold of $\alpha = 0.3$. We obtain:

```
> expert(x, "cooke", probs, true.seed,
+     alpha = 0.03)

Aggregated Distribution Using Cooke Model

         Interval Probability
 (305000, 512931]         0.1
 (512931, 563423]         0.4
 (563423, 628864]         0.4
 (628864, 845000]         0.1

 Alpha: 0.03
```

As previously explained, if the value of α is not specified in the call to `expert`, its value is obtained by optimization and the resulting aggregated distribution is different:

```
> expert(x, "cooke", probs, true.seed)

Aggregated Distribution Using Cooke Model

         Interval Probability
 (305000, 550000]         0.1
 (550000, 600000]         0.4
 (600000, 650000]         0.4
 (650000, 845000]         0.1

 Alpha: 0.3448
```

The Mendel–Sheridan model yields an entirely different result that retains many of the quantiles given by the experts. The result is a more detailed distribution. Here, we store the result in an object `fit` for later use:

```
> fit <- expert(x, "ms", probs, true.seed)
> fit

Aggregated Distribution Using
Mendel-Sheridan Model

         Interval Probability
 (305000, 350000]     0.01726
 (350000, 400000]     0.06864
 (400000, 525000]     0.06864
 (525000, 550000]     0.01726
 (550000, 600000]     0.06864
 (600000, 625000]     0.06864
 (625000, 650000]     0.53636
 (650000, 700000]     0.06864
 (700000, 800000]     0.06864
 (800000, 845000]     0.01726
```

Utility functions and methods

In addition to the main function `expert`, the package features a few functions and methods to ease manipulation and analysis of the aggregated distribution. Most are inspired by similar functions in package **actuar** (Dutang et al., 2008).

First, a `summary` method for objects of class '`expert`' provides more details on the fitted model

than the `print` method, namely the number of experts, the number of seed variables and the requested and set quantiles:

```
> summary(fit)

Call:
expert(x = x, method = "ms", probs = probs,
       true.seed = true.seed)

Aggregated Distribution Using
Mendel-Sheridan Model

         Interval Probability
 (305000, 350000]     0.01726
 (350000, 400000]     0.06864
 (400000, 525000]     0.06864
 (525000, 550000]     0.01726
 (550000, 600000]     0.06864
 (600000, 625000]     0.06864
 (625000, 650000]     0.53636
 (650000, 700000]     0.06864
 (700000, 800000]     0.06864
 (800000, 845000]     0.01726

 Number of experts: 3,
 Number of seed variables: 2
 Quantiles: 0*, 0.1, 0.5, 0.9, 1*
   (* were set)
```

Methods of `mean` and `quantile` give easy access to the mean and to the quantiles of the aggregated distribution:

```
> mean(fit)

[1] 607875

> quantile(fit)

    0%    25%    50%    75%   100%
305000 600000 625000 625000 845000
```

Of course, one may also specify one or many quantiles in a second argument:

```
> quantile(fit, c(0.1, 0.9))

   10%    90%
400000 650000
```

Moreover, the `quantile` method can return linearly interpolated quantiles by setting the `smooth` argument to `TRUE`:

```
> quantile(fit, smooth = TRUE)

    0%    25%    50%    75%   100%
305000 603478 633898 645551 845000
```

The inverse of the quantile and smoothed quantile functions are, respectively, the cumulative distribution function (cdf) and the ogive (Klugman et al., 1998; Goulet and Pigeon, 2008). In a fashion similar to `ecdf` of the base R package **stats**, the `cdf` function returns a function object to compute the cdf of the aggregated distribution at any point:

```
> F <- cdf(fit)
> knots(F)

 [1] 305000 350000 400000 525000 550000
 [6] 600000 625000 650000 700000 800000
[11] 845000

> F(knots(F))

 [1] 0.00000 0.01726 0.08590 0.15455
 [5] 0.17181 0.24045 0.30909 0.84545
 [9] 0.91410 0.98274 1.00000

> F(c(450000, 750000))

[1] 0.0859 0.9141
```

A `plot` method yields the graphic in Figure 1:

```
> plot(F)
```

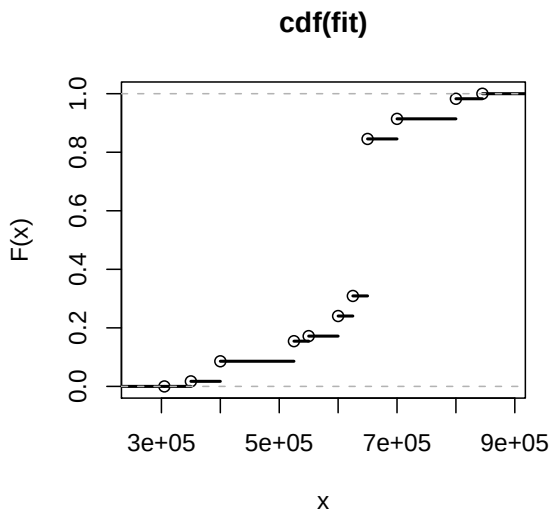


Figure 1: Cumulative distribution function of the aggregated distribution

Function `ogive` is the analogue of `cdf` for the linearly interpolated cdf:

```
> G <- ogive(fit)
> G(knots(G))

 [1] 0.00000 0.01726 0.08590 0.15455
 [5] 0.17181 0.24045 0.30909 0.84545
 [9] 0.91410 0.98274 1.00000

> G(c(450000, 750000))

[1] 0.1134 0.9484

> plot(G)
```

See Figure 2 for the graphic created with the last expression.

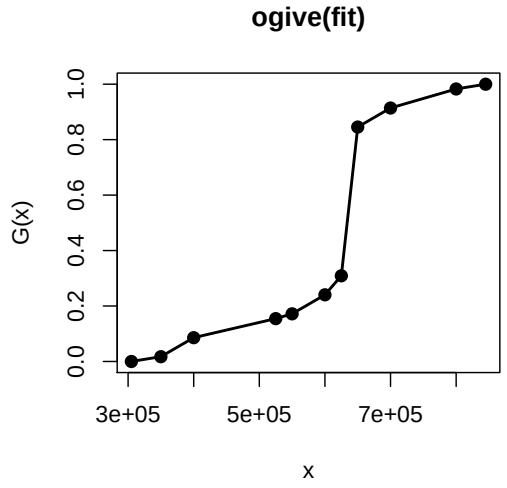


Figure 2: Ogive of the aggregated distribution

Finally, a method of `hist` draws the derivative of the ogive, that is the histogram. The arguments of the method are the same as those of the default method in package **stats**. See Figure 3 for the graphic created by the following expression:

```
> hist(fit)
```

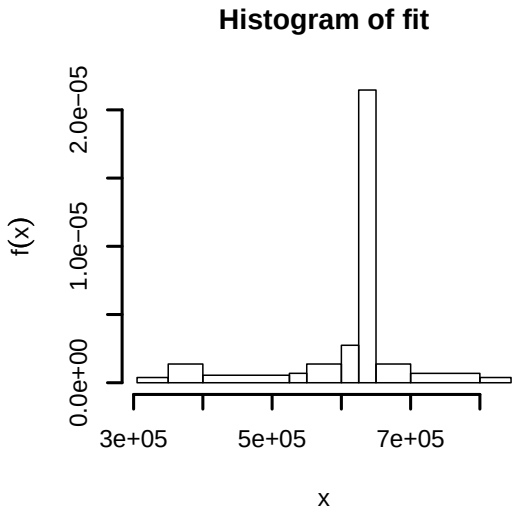


Figure 3: Histogram of the aggregated distribution

Further model fitting

Depending on the application, the aggregated distribution one obtains from `expert` might be too crude for risk analysis, especially if one used Cooke's model with very few quantiles. Now, the expert aggregated distribution actually plays the role of the empirical distribution in usual statistical modeling. That means one can further fit a continuous or discrete distribution to the aggregated "data".

For example, consider the task of fitting a gamma distribution to the Mendel–Sheridan aggregated distribution obtained above. Among many other options (see, e.g., Klugman et al., 1998; Goulet and Pigeon, 2008), we choose our parameters to minimize the Cramér-von Mises distance

$$d(\alpha,\lambda) = \sum_{i=1}^{n} (G(x_i) - F(x_i;\alpha,\lambda))^2,$$

where $G(\cdot)$ is the ogive of the aggregated distribution, $x_1,\ldots,x_n$ are its knots and $F(\cdot;\alpha,\lambda)$ is the cdf of a gamma distribution with shape parameter α and rate parameter λ. The optimal parameters are simple to obtain with the function `optim`:

```
> f <- function(p, x) drop(crossprod(G(x) -
+     pgamma(x, p[1], p[2])))
> optim(c(100, 0.00016), f,
        x = knots(G))$par

[1] 4.233e+02 6.716e-04
```

Conclusion

The paper introduced a package to elicit statistical distributions from expert opinion in situations where real data is scarce or absent. The main function, `expert`, is a unified interface to the two most important models in the field — the Cooke classical model and the Mendel–Sheridan Bayesian model — as well as the informal predefined weights model. The package also provides functions and methods to compute from the object returned by `expert` and plot important results. Most notably, `cdf`, `ogive` and `quantile` give access to the cumulative distribution function of the aggregated distribution, its ogive and the inverse of the cdf and ogive.

For modeling purposes, the package can be used as a direct replacement of Excalibur. However, coupled with the outstanding statistical and graphical capabilities of R, we feel **expert** goes far beyond Excalibur in terms of usability and interoperability.

Acknowledgments

This research benefited from financial support from the Natural Sciences and Engineering Research Council of Canada and from the *Chaire d'actuariat* (Actuarial Science Chair) of Université Laval. We also thank one anonymous referee and the R Journal editors for corrections and improvements to the paper.

Bibliography

R. Cooke. *Experts in Uncertainty*. Oxford University Press, 1991.

R. Cooke and L. Goossens. TU Delft expert judgment data base. *Reliability Engineering and System Safety*, 93(5):657–674, 2008.

C. Dutang, V. Goulet, and M. Pigeon. **actuar**: An R package for actuarial science. *Journal of Statistical Software*, 25(7), 2008. URL http://www.jstatsoft.org/v25/i07.

V. Goulet and M. Pigeon. Statistical modeling of loss distributions using **actuar**. *R News*, 8(1):34–40, May 2008. URL http://cran.r-project.org/doc/Rnews/Rnews_2008-1.pdf.

V. Goulet, M. Jacques, and M. Pigeon. Modeling of data using expert opinion: An actuarial perspective. 2008. In preparation.

J. B. Kadane and R. L. Winkler. Separating probability elicitation from utilities. *Journal of the American Statistical Association*, 83(402):357–363, 1988.

J. B. Kadane and L. J. Wolfson. Experiences in elicitation. *The Statistician*, 47(1):3–19, 1998.

S. A. Klugman, H. H. Panjer, and G. Willmot. *Loss Models: From Data to Decisions*. Wiley, New York, 1998. ISBN 0-4712388-4-8.

M. Mendel and T. Sheridan. Filtering information from human experts. *IEEE Transactions on Systems, Man and Cybernetics*, 36(1):6–16, 1989.

A. O'Hagan, C. Buck, C. Daneshkhah, J. Eiser, P. Garthwaite, D. Jenkinson, J. Oakley, and T. Rakow. *Uncertain Judgements*. Wiley, 2006.

F. Ouchi. A literature review on the use of expert opinion in probabilistic risk analysis. Policy Research Working Paper 3201, World Bank, 2004. URL http://econ.worldbank.org/.

M. Pigeon. Utilisation d'avis d'experts en actuariat. Master's thesis, Université Laval, 2008.

A. Tversky and D. Kahneman. Judgment under uncertainty: Heuristics and biases. *Science, New Series*, 185(4157):1124–1131, 1974.

Vincent Goulet
École d'actuariat
Université Laval
Québec, Canada
vincent.goulet@act.ulaval.ca

Michel Jacques
École d'actuariat
Université Laval
Québec, Canada
michel.jacques@act.ulaval.ca

Mathieu Pigeon
Institut de Statistique
Université Catholique de Louvain
Louvain-la-Neuve, Belgium
mathieu.pigeon@uclouvain.be

mvtnorm: New Numerical Algorithm for Multivariate Normal Probabilities

by Xuefei Mi, Tetsuhisa Miwa and Torsten Hothorn

Miwa et al. (2003) proposed a numerical algorithm for evaluating multivariate normal probabilities. Starting with version 0.9-0 of the **mvtnorm** package (Hothorn et al., 2001; Genz et al., 2008), this algorithm is available to the R community. We give a brief introduction to Miwa's procedure and compare it, with respect to computing time and accuracy, to a quasi-randomized Monte-Carlo procedure proposed by Genz and Bretz (1999), which has been available through **mvtnorm** for some years now.

The new algorithm is applicable to problems with dimension smaller than 20, whereas the procedures by Genz and Bretz (1999) can be used to evaluate 1000-dimensional normal distributions. At the end of this article, a suggestion is given for choosing a suitable algorithm in different situations.

Introduction

An algorithm for calculating any non-centered orthant probability of a non-singular multivariate normal distribution is described by Miwa et al. (2003). The probability function in a one-sided problem is

$$\begin{aligned} P_m(\mu, \mathbf{R}) &= \mathbb{P}\{X_i \geq 0; 1 \leq i \leq m\} \\ &= \int_0^\infty \dots \int_0^\infty \phi_m(\mathbf{x}; \mu, \mathbf{R})\, dx_1 \dots dx_m \end{aligned}$$

where $\mu = (\mu_1, ..., \mu_m)^\top$ is the mean and and $\mathbf{R} = (\rho_{ij})$ the correlation matrix of m multivariate normal distributed random variables $X_1, \dots, X_m \sim \mathcal{N}_m(\mu, \mathbf{R})$. The function ϕ_m denotes the density function of the m-dimensional normal distribution with mean μ and correlation matrix $\mathbf{R}$.

The distribution function for $\mathbf{c} = (c_1, \dots, c_m)$ can be expressed as:

$$\begin{aligned} F_m(\mathbf{c}) &= \mathbb{P}\{X_i \leq c_i; 1 \leq i \leq m\} \\ &= \mathbb{P}\{-X_i \geq -c_i; 1 \leq i \leq m\} \\ &= P_m(-\mu + \mathbf{c}, \mathbf{R}). \end{aligned}$$

The m-dimensional non-centered orthant one-sided probability can be calculated from at most $(m-1)!$ non-centered probabilities with positive definite tridiagonal correlation matrix. The algorithm for calculating such probabilities is a recursive linear integration procedure. The total order of a one-sided problem is $G \times m!$, where G is the number of grid points for integration.

The two-sided probability

$$F_m(\mathbf{d}, \mathbf{c}) = \mathbb{P}\{d_i \leq X_i \leq c_i; 1 \leq i \leq m\}$$

can be calculated from 2^m m-dimensional one-sided probabilities which have the same mean and correlation matrix. The total order of this two-sided problem is $G \times m! \times 2^m$.

A new `algorithm` argument to `pmvnorm()` and `qmvnorm()` has been introduced in **mvtnorm** version 0.9-0 in order to switch between two algorithms: `GenzBretz()` is the default and triggers the use of the above mentioned quasi-randomized Monte-Carlo procedure by Genz and Bretz (1999). Alternatively, `algorithm = Miwa()` applies the procedure discussed here. Both functions can be used to specify hyper-parameters of the algorithm. For `Miwa()`, the argument `steps` defines the number of grid points G to be evaluated.

The following example shows how to calculate the probability

$$\begin{aligned} p &= F_m(\mathbf{d}, \mathbf{c}) \\ &= \mathbb{P}\{-1 < X_1 < 1, -4 < X_2 < 4, -2 < X_3 < 2\}. \end{aligned}$$

with mean $\mu = (0,0,0)^\top$ and correlation matrix

$$\mathbf{R} = \begin{pmatrix} 1 & 1/4 & 1/5 \\ 1/4 & 1 & 1/3 \\ 1/5 & 1/3 & 1 \end{pmatrix}$$

by using the following R code:

```
> library("mvtnorm")
> m <- 3
> S <- diag(m)
> S[2, 1] <- S[1, 2] <- 1 / 4
> S[3, 1] <- S[3, 1] <- 1 / 5
> S[3, 2] <- S[3, 2] <- 1 / 3
> pmvnorm(lower = -c(1,4,2),
+         upper = c(1,4,2),
+         mean=rep(0, m), sigma = S,
+         algorithm = Miwa())

[1] 0.6536804
attr(,"error")
[1] NA
attr(,"msg")
[1] "Normal Completion"
```

The upper limit and lower limit of the integral region are passed by the vectors `upper` and `lower`. The mean vector and correlation matrix are given by the vector `mean` and the matrix `sigma`. From the result we know that $p = 0.6536804$ with given correlation matrix $\mathbf{R}$.

Accuracy and time consumption

In this section we compare the accuracy and time consumption of the R implementation of the algo-

Algorithm	$m=5$		$m=10$	
	$\rho=\frac{1}{2}$	$\rho=-\frac{1}{2}$	$\rho=\frac{1}{2}$	$\rho=-\frac{1}{2}$
Genz & Bretz ($\varepsilon=10^{-4}$)	0.08468833	0.001385620	0.008863600	2.376316×10^{-8}
Genz & Bretz ($\varepsilon=10^{-5}$)	0.08472561	**0.001390769**	0.008863877	2.319286×10^{-8}
Genz & Bretz ($\varepsilon=10^{-6}$)	**0.08472682**	0.001388424	**0.008862195**	2.671923×10^{-8}
Miwa ($G=128$)	0.08472222	0.001388889	0.008863235	2.505205×10^{-8}
Exact.	0.08472222	0.001388889	0.008863236	2.505211×10^{-8}

Table 1: Value of probabilities with tri-diagonal correlation coefficients, $\rho_{i,i\pm1}=2^{-1}, 1\le i\le m$ and $\rho_{i,j}=0, \forall|i-j|>1$.

rithm of Miwa et al. (2003) with the default procedure for approximating multivariate normal probabilities in **mvtnorm** by Genz and Bretz (1999). The experiments were performed using an Intel® Pentium® 4 processor with 2.8 GHz.

Probabilities with tri-diagonal correlation matrix

The exact value of $P_m(\mu,\mathbf{R})$ is known if $\mathbf{R}$ has some special structure. E.g., when $\mathbf{R}$ is a m-dimensional tri-diagonal correlation matrix with correlation coefficients

$$\rho_{i,j}=\begin{cases} 2^{-1} & j=i\pm1 \\ 0 & |i-j|>1 \end{cases} \quad 1\le i\le m$$

the value of $P_m(\mathbf{0},\mathbf{R})$ is $((1+m)!)^{-1}$ (Miwa et al., 2003). The accuracy of Miwa' algorithm ($G=128$ grid points) and the Genz & Bretz algorithm (with absolute error tolerance $\varepsilon=10^{-4},10^{-5},10^{-6}$) for probabilities with tri-diagonal correlation matrix are compared in Table 1. In each calculation we have results with small variance. The values which do not hold the tolerance error are marked with bold characters in the tables. When the dimension is larger than five, using Genz & Bretz' algorithm, it is hard to achieve an error tolerance smaller than 10^{-5}.

Orthant probabilities

When $\mathbf{R}$ is the correlation matrix with $\rho_{i,j}=2^{-1}, i\neq j$, the value of $P_m(\mathbf{0},\mathbf{R})$ is known to be $(1+m)^{-1}$ (Miwa et al., 2003). Accuracy and time consumption of Miwa's algorithm and Genz & Bretz' algorithm for this situation are compared in Table 2.

Time consumption

Miwa's algorithm is a numerical method which has advantages in achieving higher accuracy with less time consumption compared to the stochastic method. However, Miwa's algorithm has some disadvantages. When the dimension m increases, the time consumption of Miwa's algorithm increases dramatically. Moreover, it can not be applied to singular problems which are common in multiple testing problems, for example.

Conclusion

We have implemented an R interface to the procedure of Miwa et al. (2003) in the R package **mvtnorm**. For small dimensions, it is an alternative to quasi-randomized Monte-Carlo procedures which are used by default. For larger dimensions and especially for singular problems, the method is not applicable.

Bibliography

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.

Algorithm	m=5		m=9	
	$\rho=\frac{1}{2}$	sec.	$\rho=\frac{1}{2}$	sec.
Genz & Bretz ($\varepsilon=10^{-4}$)	0.1666398	0.029	0.09998728	0.231
Genz & Bretz ($\varepsilon=10^{-5}$)	0.1666719	0.132	0.09998277	0.403
Genz & Bretz ($\varepsilon=10^{-6}$)	0.1666686	0.133	0.09999726	0.431
Miwa ($G=128$)	0.1666667	0.021	0.09999995	89.921
Exact.	0.1666667		0.10000000	

Table 2: Accuracy and time consumption of centered orthant probabilities with correlation coefficients, $\rho_{i,j}=2^{-1}, i\neq j$

Dimension	Miwa ($G = 128$)		Genz & Bretz ($\varepsilon = 10^{-4}$)	
	Single	Double	Single	Double
$m = 5$	0.021	0.441	0.029	0.085
$m = 6$	0.089	8.731	0.089	0.149
$m = 7$	0.599	156.01	0.083	0.255
$m = 8$	9.956	4hours	0.138	0.233
$m = 9$	89.921	-	0.231	0.392

Table 3: Time consumption of non-centered orthant probabilities (measured in seconds).

A. Genz, F. Bretz, T. Hothorn, T. Miwa, X. Mi, F. Leisch, and F. Scheipl. *mvtnorm: Multivariate Normal and T Distribution*, 2008. URL http://CRAN.R-project.org. R package version 0.9-0.

T. Hothorn, F. Bretz, and A. Genz. On multivariate t and Gauß probabilities in R. *R News*, 1(2):27–29, June 2001.

T. Miwa, A. J. Hayter, and S. Kuriki. The evaluation of general non-centred orthant probabilities. *Journal of The Royal Statistical Society Series B–Statistical Methodology*, 65:223–234, 2003.

Xuefei Mi
Institut für Biostatistik
Leibniz Universität Hannover, Germany
mi@biostat.uni-hannover.de

Tetsuhisa Miwa
National Institute for Agro-Environmental Sciences
Kannondai, Japan
miwa@niaes.affrc.go.jp

Torsten Hothorn
Institut für Statistik
Ludwig-Maximilians-Universität München, Germany
Torsten.Hothorn@R-project.org

EMD: A Package for Empirical Mode Decomposition and Hilbert Spectrum

by Donghoh Kim and Hee-Seok Oh

Introduction

The concept of empirical mode decomposition (EMD) and the Hilbert spectrum (HS) has been developed rapidly in many disciplines of science and engineering since Huang et al. (1998) invented EMD. The key feature of EMD is to decompose a signal into so-called intrinsic mode function (IMF). Furthermore, the Hilbert spectral analysis of intrinsic mode functions provides frequency information evolving with time and quantifies the amount of variation due to oscillation at different time scales and time locations. In this article, we introduce an R package called **EMD** (Kim and Oh, 2008) that performs one- and two- dimensional EMD and HS.

Intrinsic mode function

The essential step extracting an IMF is to identify an oscillation embedded in a signal from local time scale. Consider the following synthetic signal $x(t), 0 < t < 9$ of the form

$$x(t) = 0.5t + \sin(\pi t) + \sin(2\pi t) + \sin(6\pi t). \quad (1)$$

The signal in Figure 1 consists of several components, which are generated through the process that a component is superimposed to each other.

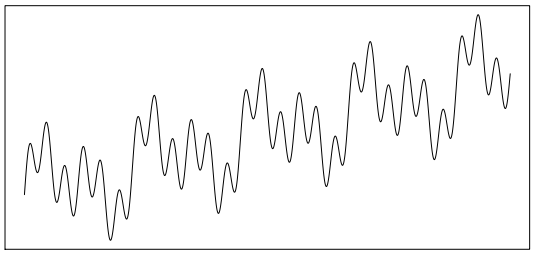


Figure 1: A sinusoidal function having 4 components

An intrinsic oscillation or frequency of a component, for example, $\sin(\pi t), t \in (0,9)$ in Figure 1 can be perceived through the red solid wave or the blue dotted wave in Figure 2. The blue dotted wave in Figure 2 illustrates one cycle of intrinsic oscillation which starts at a local maximum and terminates at a consecutive local maximum by passing through two zeros and a local minimum which eventually appears between two consecutive maxima. A component for a given time scale can be regarded as the composition of repeated intrinsic oscillation which is symmetric to its local mean, zero.

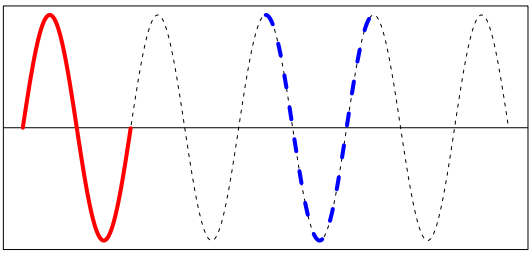


Figure 2: A sinusoidal function

Thus the first step to define intrinsic oscillation is to detect local extrema or zero-crossings. The function `extrema()` identifies local extrema and zero-crossings of the signal in Figure 2.

```
> ### Identify extrema and zero-crossings
> ndata <- 3000
> tt <- seq(0, 9, length=ndata)
> xt <- sin(pi * tt)
>
> library(EMD)
> extrema(xt)
$minindex
     [,1] [,2]
[1,]  501  501
[2,] 1167 1167
[3,] 1834 1834
[4,] 2500 2500

$maxindex
     [,1] [,2]
[1,]  168  168
[2,]  834  834
[3,] 1500 1501
[4,] 2167 2167
[5,] 2833 2833

$nextreme
[1] 9

$cross
      [,1] [,2]
 [1,]    1    1
 [2,]  334  335
 [3,]  667  668
 [4,] 1000 1001
 [5,] 1333 1334
 [6,] 1667 1668
 [7,] 2000 2001
 [8,] 2333 2334
 [9,] 2666 2667

$ncross
[1] 9
```

The function `extrema()` returns a list of followings.

- `minindex` : matrix of time index at which local minima are attained. Each row specifies a starting and ending time index of a local minimum.
- `maxindex` : matrix of time index at which local maxima are attained. Each row specifies a starting and ending time index of a local maximum.
- `nextreme` : the number of extrema.
- `cross` : matrix of time index of zero-crossings. Each row specifies a starting and ending time index of zero-crossings.
- `ncross` : the number of zero-crossings.

Once local extrema is obtained, the intrinsic mode function is derived through the sifting procedure.

Sifting process

Huang et al. (1998) suggested a data-adapted algorithm extracting a sinusoidal wave or equivalently a frequency from a given signal x. First, identify the local extrema in Figure 3(a), and generate the two functions called the upper envelope and lower envelope by interpolating local maxima and local minima, respectively. See Figure 3(b). Second, take their average, which will produce a lower frequency component than the original signal as in Figure 3(c). Third, by subtracting the envelope mean from the signal x, the highly oscillated pattern h is separated as in Figure 3(d).

Huang et al. (1998) defined an oscillating wave as an intrinsic mode function if it satisfies two conditions 1) the number of extrema and the number of zero-crossings differs only by one and 2) the local average is zero. If the conditions of IMF are not satisfied after one iteration of aforementioned procedure, the same procedure is applied to the residue signal as in Figure 3(d), (e) and (f) until properties of IMF are satisfied.

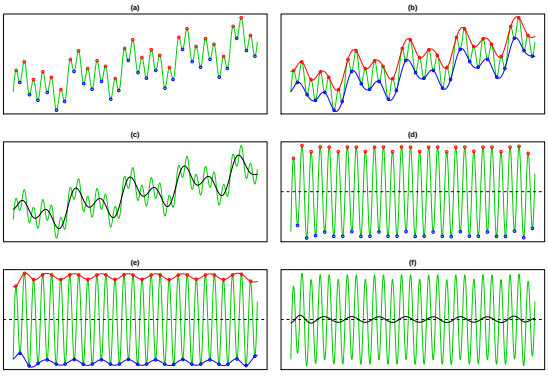


Figure 3: Sifting procedure

This iterative process is called sifting. The following code produces Figure 3, and the function `extractimf()` implements the sifting algorithm by identifying the local extrema with the `extrema()`. Note that when setting the option `'check=TRUE'`, one must click the plot to proceed to the next step.

```
> ### Generating a signal
> ndata <- 3000
> par(mfrow=c(1,1), mar=c(1,1,1,1))
> tt2 <- seq(0, 9, length=ndata)
> xt2 <- sin(pi * tt2) + sin(2* pi * tt2) +
+ sin(6 * pi * tt2)  + 0.5 * tt2
> plot(tt2, xt2, xlab="", ylab="", type="l",
+ axes=FALSE); box()
>
> ### Extracting the first IMF by sifting process
> tryimf <- extractimf(xt2, tt2, check=TRUE)
```

The function `extractimf()` extracts IMF's from a given signal, and it is controlled by the following arguments.

- `residue` : observation or signal observed at time `tt`.
- `tt` : observation index or time index.
- `tol` : tolerance for stopping rule.
- `max.sift` : the maximum number of sifting.
- `stoprule` : stopping rule.
- `boundary` : specifies boundary condition.
- `check` : specifies whether the sifting process is displayed. If `check=TRUE`, click the plotting area to start the next step.

Stopping rule

The sifting process stops when the replication of sifting procedure exceed the predefined maximum number by `max.sift` or satisfies the properties of IMF by stopping rule. The stopping rule `stoprule` has two options – `"type1"` and `"type2"`. The option `stoprule = "type1"` makes the sifting process stop when the absolute values of the candidate IMF h_i are smaller than tolerance level, that is, $|h_i(t)| <$ `tol` for all t. Or by the option `stoprule = "type2"`, the sifting process stops when the variation of consecutive candidate IMF's is within the tolerance level,

$$\sum_t \left(\frac{h_i(t) - h_{i-1}(t)}{h_{i-1}(t)} \right)^2 < \texttt{tol}.$$

Boundary adjustment

To eliminate the boundary effect of a signal, it is necessary to adjust a signal at the boundary. Huang et al. (1998) extended the original signal by adding artificial waves repeatedly on both sides of the boundaries. The waves called characteristic waves are constructed by repeating the implicit mode formed from extreme values nearest to boundary. The argument `boundary` specifies the adjusting method of the

boundary. The argument `boundary = "wave"` constructs a wave which is defined by two consecutive extrema at either boundary, and adds four waves at either end. Typical adjusting method extends a signal assuming that a signal is symmetric or periodic. The option `boundary = "symmetric"` or `boundary = "periodic"` extends both boundaries symmetrically or periodically.

Zeng and He (2004) considered two extended signals by adding a signal in a symmetric way and reflexive way called even extension and odd extension, respectively. Even extension and odd extension produce the extended signals so that its average is zero. This boundary condition can be specified by `boundary = "evenodd"`. For each extended signal, upper and lower envelopes are constructed and envelope mean of the extended signals is defined by the average of four envelopes. Then, the envelope mean outside the time scale of the original signal is close to zero, while the envelope mean within the time scale of the original signal is almost the same as the envelope mean of the original signal. On the other hand, the option `boundary = "none"` performs no boundary adjustments.

Empirical mode decomposition

Once the highest frequency is removed from a signal, the same procedure is applied on the residue signal to identify next highest frequency. The residue is considered a new signal to decompose.

Suppose that we have a signal from model (1). The signal in Figure 1 is composed of 4 components from $\sin(6\pi t)$ with the highest frequency to $0.5t$ with the lowest frequency. We may regard the linear component as a component having the lowest frequency. The left panel in Figure 4 illustrates the first IMF and the residue signal obtained by the function `extractimf()`. If the remaining signal is still compound of components with several frequencies as in the left panel in Figure 4, then the next IMF is obtained by taking the residue signal as a new signal in the right panel in Figure 4. The number of extrema will decrease as the procedure continues, so that the signal is sequently decomposed into the highest frequency component imf_1 to the lowest frequency component imf_n, for some finite n and a residue signal r. Finally, we have n IMF's and a residue signal as

$$x(t) = \sum_{i=1}^{n} imf_i(t) + r(t).$$

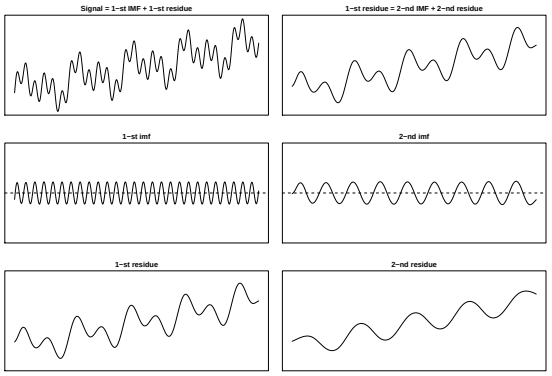


Figure 4: Two IMF's by the sifting algorithm

The above-mentioned decomposition process is implemented by the function `emd()` that utilizes the functions `extractimf()` and `extrema()`. The final decomposition result by the following code is illustrated in Figure 5.

```
> ### Empirical Mode Decomposition
> par(mfrow=c(3,1), mar=c(2,1,2,1))
> try <- emd(xt2, tt2, boundary="wave")
>
> ### Ploting the IMF's
> par(mfrow=c(3,1), mar=c(2,1,2,1))
> par(mfrow=c(try$nimf+1, 1), mar=c(2,1,2,1))
> rangeimf <- range(try$imf)
> for(i in 1:try$nimf)
+ plot(tt2, try$imf[,i], type="l", xlab="",
+ ylab="", ylim=rangeimf, main=
+ paste(i, "-th IMF", sep="")); abline(h=0)
> plot(tt2, try$residue, xlab="", ylab="",
+ main="residue", type="l")
```

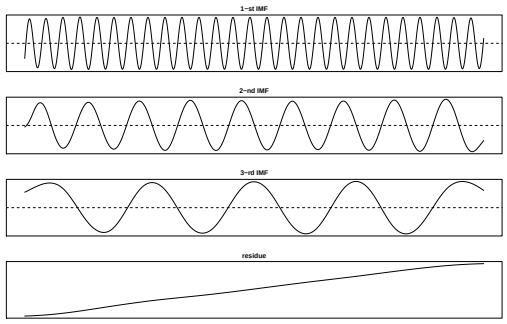


Figure 5: Decomposition of a signal by model (1)

The arguments of `emd()` are similar to those of `extractimf()`. The additional arguments are

- `max.imf` : the maximum number of IMF's.
- `plot.imf` : specifies whether each IMF is displayed. If `plot.imf=TRUE`, click the plotting area to start the next step.

Up to now we have focused on artificial signals without any measurement error. A typical signal in the real world is corrupted by noise, which is not the component of interest and contains no interpretable

information. A remedy to smooth out the noise is to apply smoothing technique not interpolation during the sifting process. Then the first IMF might capture the entire noise effectively. As an alternative, Kim and Oh (Kim and Oh, 2006) proposed an efficient smoothing method for IMF's by combining the conventional cross-validation and thresholding approach. By thresholding, noisy signal can be denoised while the distinct localized feature of a signal can be kept.

Intermittence

Huang et al. (1998, 2003) pointed out that intermittence raises mode mixing, which means that different modes of oscillations coexist in a single IMF. Since EMD traces the highest frequency embedded in a given signal locally, when intermittence occurs, the shape of resulting IMF is abruptly changed and this effect distorts procedures thereafter.

Huang et al. (2003) attacked this phenomenon by restricting the size of frequency. To be specific, the distance limit of the successive maxima (minima) in an IMF is introduced. Thus, IMF composes of only sinusoidal waves whose length of successive maxima (minima) are shorter than their limit. Equivalently, we may employ the length of the zero-crossings to overcome the intermittence problem. Consider a signal $x(t)$ combined by two sine curves (Deering and Kaiser, 2005),

$$x(t) = \begin{cases} \sin(2\pi f_1 t) + \sin(2\pi f_2 t), & \frac{1}{30} \le t \le \frac{2}{30}, \\ \sin(2\pi f_1 t), & \text{otherwise.} \end{cases} \quad (2)$$

Figure 6 illustrates the signal $x(t)$ when $f_1 = 1776$ and $f_2 = 1000$ and the corresponding two IMF's. The first IMF absorbs the component that appeared in the second IMF between $\frac{1}{30}$ and $\frac{2}{30}$. Thus, the resulting IMF has a mode mixing pattern.

```
> ### Mode mixing
> tt <- seq(0, 0.1, length = 2001)[1:2000]
> f1 <- 1776; f2 <- 1000
> xt <- sin(2*pi*f1*tt) * (tt <= 0.033 |
+ tt >= 0.067) + sin(2*pi*f2*tt)
>
> ### EMD
> interm1 <- emd(xt, tt, boundary="wave",
+ max.imf=2, plot.imf=FALSE)
> par(mfrow=c(3, 1), mar=c(3,2,2,1))
> plot(tt, xt, main="Signal", type="l")
> rangeimf <- range(interm1$imf)
> plot(tt, interm1$imf[,1], type="l", xlab="",
+ ylab="", ylim=rangeimf, main="IMF 1")
> plot(tt, interm1$imf[,2], type="l", xlab="",
+ ylab="", ylim=rangeimf, main="IMF 2")
```

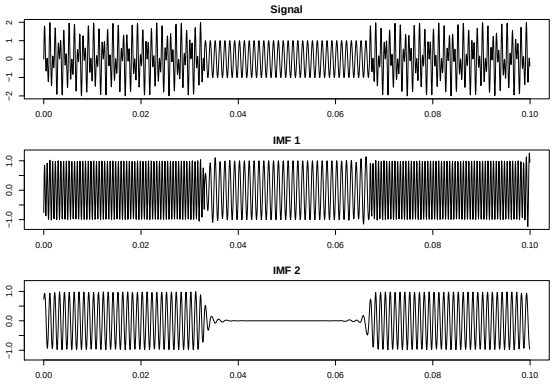


Figure 6: Signal $x(t)$ by model (2) and first two IMF's

By following the approach of Huang et al. (1998, 2003), we can remove waves whose empirical period represented by the distance of other zero-crossings is larger than 0.0007 in the first IMF. The period information obtained by histogram in Figure 7 can be used to choose an appropriate distance. We eliminate the waves with lower frequency in the first IMF with the histogram of other zero-crossings.

```
> ### Histogram of empirical period
> par(mfrow=c(1,1), mar=c(2,4,1,1))
> tmpinterm <- extrema(interm1$imf[,1])
> zerocross <-
+ as.numeric(round(apply(tmpinterm$cross, 1, mean)))
> hist(diff(tt[zerocross[seq(1, length(zerocross),
+ by=2)]]), freq=FALSE, xlab="", main="")
```

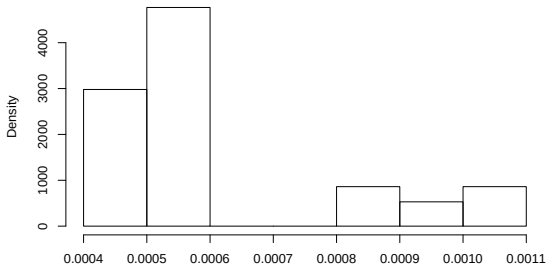


Figure 7: Histogram of the empirical period

Figure 8 shows the resulting IMF's after treating intermittence properly. The argument `interm` of the function `emd()` specifies a vector of periods to be excluded from the IMF's.

```
> ### Treating intermittence
> interm2 <- emd(xt, tt, boundary="wave",
+ max.imf=2, plot.imf=FALSE, interm=0.0007)
>
> ### Plot of each imf
> par(mfrow=c(2,1), mar=c(2,2,3,1), oma=c(0,0,0,0))
> rangeimf <- range(interm2$imf)
> plot(tt,interm2$imf[,1], type="l",
+ main="IMF 1 after treating intermittence",
+ xlab="", ylab="", ylim=rangeimf)
> plot(tt,interm2$imf[,2], type="l",
+ main="IMF 2 after treating intermittence",
+ xlab="", ylab="", ylim=rangeimf)
```

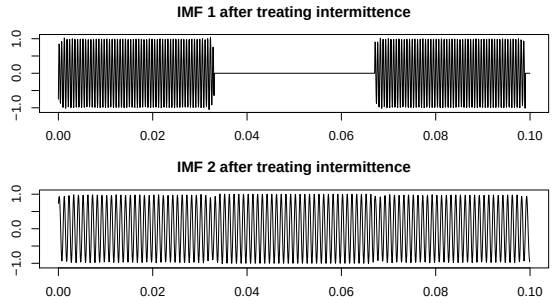


Figure 8: Decomposition of signal $x(t)$ by model (2) after treating the intermittence.

Hilbert spectrum

When a signal is subject to non-stationarity so that the frequency and amplitude change over time, it is necessary to have a more flexible and extended notion of frequency. Huang et al. (1998) used the concept of instantaneous frequency through the Hilbert transform. For a comprehensive explanation of the Hilbert transform, refer to Cohen (1995). For a real signal $x(t)$, the analytic signal $z(t)$ is defined as $z(t) = x(t) + i\, y(t)$ where $y(t)$ is the Hilbert transform of $x(t)$, that is, $y(t) = \frac{1}{\pi} P \int_{-\infty}^{\infty} \frac{x(s)}{t-s}\, ds$ where P is the Cauchy principal value. The polar coordinate form of the analytic signal z with amplitude and phase is $z(t) = a(t)\exp(i\theta(t))$ where amplitude $a(t)$ is $||z(t)|| = \sqrt{x(t)^2 + y(t)^2}$ and phase $\theta(t)$ is $\arctan\left(\frac{y(t)}{x(t)}\right)$. The instantaneous frequency as time-varying phase is defined as $\frac{d\theta(t)}{dt}$. After decomposing a signal into IMF's with EMD thereby preserving any local property in the time domain, we can extract localized information in the frequency domain with the Hilbert transform and identify hidden local structures embedded in the original signal. The local information can be described by the Hilbert spectrum which is amplitude and instantaneous frequency representation with respect to time. Figure 9 describes the Hilbert spectrum for IMF 1 of the signal of model (2) before and after treating the intermittence. The X-Y axis represents time and instantaneous frequency, and the color intensity of the image depicts instantaneous amplitude.

```
> ### Spectrogram : X - Time, Y - frequency,
> ### Z (Image) - Amplitude
> test1 <- hilbertspec(interm1$imf)
> spectrogram(test1$amplitude[,1],
+ test1$instantfreq[,1])
> test2 <- hilbertspec(interm2$imf, tt=tt)
> spectrogram(test2$amplitude[,1],
+ test2$instantfreq[,1])
```

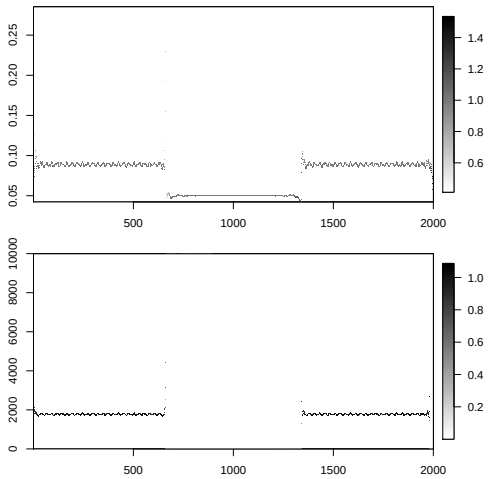


Figure 9: The Hilbert spectrum for IMF 1 of the signal of model (2)

For multiple signals, the function `hilbertspec()` calculates the amplitudes and instantaneous frequency using Hilbert transform. The function has the following arguments,

- `xt` : matrix of multiple signals. Each column represents a signal.
- `tt` : observation index or time index.

The function `hilbertspec()` returns a matrix of amplitudes and instantaneous frequencies for multiple signals. The function `spectrogram()` produces an image of amplitude by time index and instantaneous frequency. The horizontal axis represents time, the vertical axis is instantaneous frequency, and the color of each point in the image represents amplitude of a particular frequency at a particular time. It has arguments as

- `amplitude` : vector or matrix of amplitudes for multiple signals.
- `freq` : vector or matrix of instantaneous frequencies for multiple signals.
- `tt` : observation index or time index.
- `multi` : specifies whether spectrograms of multiple signals are separated or not.
- `nlevel` : the number of color levels used in legend strip
- `size` : vector of image size.

Extension to two dimensional image

The extension of EMD to an image or two dimensional data is straightforward except the identification of the local extrema. Once the local extrema

are identified, the two dimensional smoothing spline technique is used for the sifting procedure.

For the two-dimensional case, we provide four R functions.

(1) `extrema2dC()` for identifying the two dimensional extrema,

(2) `extractimf2d()` for extracting the IMF from a given image,

(3) `emd2d()` for decomposing an image to IMF's and the residue image combining two R functions above, and

(4) `imageEMD()` for displaying the decomposition results.

As in a one-dimensional case, `extractimf2d()` extracts two dimensional IMF's from a given image based on local extrema identified by `extrema2dC()`. Combining these functions, `emd2d()` performs decomposition and its arguments are as follows.

- `z` : matrix of an image observed at (`x`, `y`).
- `x, y` : locations of regular grid at which the values in `z` are measured.
- `tol` : tolerance for stopping rule of sifting.
- `max.sift` : the maximum number of sifting.
- `boundary` : specifies boundary condition 'symmetric', 'reflexive' or 'none'.
- `boundperc` : expand an image by adding specified percentage of image at the boundary when boundary condition is 'symmetric' or 'reflexive'.
- `max.imf` : the maximum number of IMF.
- `plot.imf` : specifies whether each IMF is displayed. If `plot.imf=TRUE`, click the plotting area to start the next step.

The following R code performs two dimensional EMD of the Lena image. The size of the original image is reduced for computational simplicity.

```
> data(lena)
> z <- lena[seq(1, 512, by=4), seq(1, 512, by=4)]
> lenadecom <- emd2d(z, max.imf = 4)
```

The R function `imageEMD()` plots decomposition results and the argument `extrma=TRUE` illustrates the local maxima (minima) with the white (black) color and grey background. See Figure 10.

```
> imageEMD(z=z, emdz=lenadecom, extrema=TRUE,
+ col=gray(0:100/100))
```

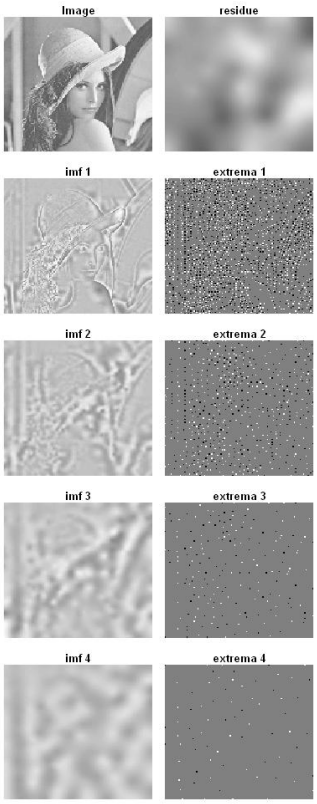


Figure 10: Decomposition of the Lena image

Conclusions

IMF's through EMD provide a multi-resolution tool and spectral analysis gives local information with time-varying amplitude and phase according to the scales. We introduce **EMD**, an R package for the proper implementation of EMD, and the Hilbert spectral analysis for non-stationary signals. It is expected that R package **EMD** makes EMD methodology practical for many statistical applications.

Bibliography

L. Cohen. *Time-Frequency Analysis*. Prentice-Hall, 1995.

R. Deering and J. F. Kaiser. The Use of a Masking Signal to Improve Empirical Mode Decomposition. *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, 4:485–488, 2005.

N. E. Huang, Z. Shen, S. R. Long, M. C. Wu, H. H. Shih, Q. Zheng, N.-C. Yen, C. C. Tung, and H. H. Liu. The Empirical Mode Decomposition and Hilbert Spectrum for Nonlinear and Nonstationary Time Series Analysis. *Proceedings of the Royal Society London A.*, 454:903–995, 1998.

N. E. Huang, M. C. Wu, S. R. Long, S. Shen, W. Qu, P. Gloerson, and K. L. Fan. A Confidence Limit for

the Empirical Mode Decomposition and Hilbert Spectral Analysis. *Proceedings of the Royal Society London A.*, 31:417–457, 2003.

D. Kim and H-S. Oh. Hierarchical Smoothing Technique by Empirical Mode Decomposition. *The Korean Journal of Applied Statistics*, 19:319–330, 2006.

D. Kim and H-S. Oh. **EMD**: Empirical Mode Decomposition and Hilbert Spectral Analysis, 2008. URL http://cran.r-project.org/web/packages/EMD/index.html.

D. Nychka. **fields**: Tools for Spatial Data, 2007. URL http://cran.r-project.org/web/packages/fields/index.html.

K. Zeng and M-X. He. A Simple Boundary Process Technique for Empirical Mode Decomposition. *Proceedings of IEEE International Geoscience and Remote Sensing Symposium*, 6:4258–4261, 2004.

Donghoh Kim
Sejong University, Korea
E-mail: donghoh.kim@gmail.com
Hee-Seok Oh
Seoul National University, Korea
E-mail: heeseok@stats.snu.ac.kr

Sample Size Estimation while Controlling False Discovery Rate for Microarray Experiments Using the ssize.fdr Package

by Megan Orr and Peng Liu

Introduction

Microarray experiments are becoming more and more popular and critical in many biological disciplines. As in any statistical experiment, appropriate experimental design is essential for reliable statistical inference, and sample size has a crucial role in experimental design. Because microarray experiments are rather costly, it is important to have an adequate sample size that will achieve a desired power without wasting resources.

For a given microarray data set, thousands of hypotheses, one for each gene, are simultaneously tested. Storey and Tibshirani (2003) argue that controlling false discovery rate (FDR) is more reasonable and more powerful than controlling family-wise error rate (FWER) in genomic data. However, the most common procedure used to calculate sample size involves controlling FWER, not FDR.

Liu and Hwang (2007) describe a method for a quick sample size calculation for microarray experiments while controlling FDR. In this paper, we introduce the R package **ssize.fdr** which implements the method proposed by Liu and Hwang (2007). This package can be applied for designing one-sample, two-sample, or multi-sample experiments. The practitioner defines the desired power, the level of FDR to control, the proportion of non-differentially expressed genes, as well as effect size and variance. More specifically, the effect size and variance can be set as fixed or random quantities coming from appropriate distributions. Based on user-defined parameters, this package creates plots of power vs. sample size. These plots allow for visualization of trade-offs between power and sample size, in addition to the changes between power and sample size when the user-defined quantities are modified.

For more in-depth details and evaluation of this sample size calculation method, please refer to Liu and Hwang (2007).

Method

For a given microarray experiment, for each gene, let $H = 0$ if the null hypothesis is true and $H = 1$ if the alternative hypothesis is true. In a microarray experiment, $H = 1$ represents differential expression for a gene, whereas $H = 0$ represents no differential expression. As in Storey and Tibshirani (2003), we assume each test is Bernoulli distributed with the probability $\Pr(H = 0) = \pi_0$, where π_0 is the proportion of non-differentially expressed genes. Liu and Hwang (2007) derived that the following must hold to control FDR at the level of α:

$$\frac{\alpha}{1-\alpha}\frac{1-\pi_0}{\pi_0} \geq \frac{\Pr(T \in \Gamma \mid H = 0)}{\Pr(T \in \Gamma \mid H = 1)} \qquad (1)$$

where T is the test statistic and Γ is the rejection region of the test. For each proposed hypothesis (test) with given user-defined quantities, the sample size is calculated using the following steps:

1. Solve for Γ using (1) for each sample size.
2. Calculate the power corresponding to Γ with appropriate formula for $\Pr(T \in \Gamma \mid H = 1)$.
3. Determine sample size based on desired power.

For specific experimental designs, the numerator and denominator of the right hand side of (1) is replaced by corresponding functions that calculate the type I error and power of the test, respectively.

Functions

The package **ssize.fdr** has six functions which includes three new functions that have recently been developed as well as three functions translated from the previous available Matlab codes. The six functions and their descriptions are listed below.

```
ssize.oneSamp(delta, sigma, fdr = 0.05,
    power = 0.8, pi0 = 0.95, maxN = 35,
    side = "two-sided", cex.title=1.15,
    cex.legend=1)

ssize.oneSampVary(deltaMean, deltaSE, a, b,
    fdr = 0.05, power = 0.8, pi0 = 0.95,
    maxN = 35, side = "two-sided",
    cex.title=1.15, cex.legend=1)

ssize.twoSamp(delta, sigma, fdr = 0.05,
     power = 0.8, pi0 = 0.95, maxN = 35,
     side = "two-sided", cex.title=1.15,
     cex.legend=1)

ssize.twoSampVary(deltaMean, deltaSE, a, b,
     fdr = 0.05, power = 0.8, pi0 = 0.95,
     maxN = 35, side = "two-sided",
     cex.title=1.15, cex.legend=1)
```

```
ssize.F(X, beta, L = NULL, dn, sigma,
     fdr = 0.05, power = 0.8, pi0 = 0.95,
     maxN = 35, cex.title=1.15,
     cex.legend=1)

ssize.Fvary(X, beta, L = NULL, dn, a, b,
     fdr = 0.05, power = 0.8, pi0 = 0.95,
     maxN = 35, cex.title=1.15,
     cex.legend=1)
```

`ssize.oneSamp` and `ssize.twoSamp` compute appropriate sample sizes for one- and two-sample microarray experiments, respectively, for fixed effect size (`delta`) and standard deviation (`sigma`). For one-sample designs, the effect size is defined as the difference in the true mean expression level and its proposed value under the null hypothesis for each gene. For two-sample designs, the effect size is defined as the difference in mean expression levels between treatment groups for each gene. In the two-sample case, `sigma` is the pooled standard deviation of treatment expression levels.

`ssize.oneSampVary` and `ssize.twoSampVary` compute appropriate sample sizes for one- and two-sample microarray experiments, respectively, in which effect sizes and standard deviations vary among genes. Effect sizes among genes are assumed to follow a normal distribution with mean `deltaMean` and standard deviation `deltaSE`, while variances (the square of the standard deviations) among genes are assumed to follow an Inverse Gamma distribution with shape parameter `a` and scale parameter `b`.

`ssize.F` computes appropriate sample sizes for multi-sample microarray experiments. Additional inputs include the design matrix (`X`), the parameter vector (`beta`), the coefficient matrix for linear contrasts of interest (`L`), a function of n for the degrees of freedom of the experimental design (`dn`), and the pooled standard deviation of treatment expression levels (`sigma`), assumed to be identical for all genes.

`ssize.Fvary` computes appropriate sample sizes for multi-sample microarray experiments in which the parameter vector is fixed for all genes, but the variances are assumed to vary among genes and follow an Inverse Gamma distribution with shape parameter `a` and scale parameter `b`.

All functions contain a default value for π_0 (`pi0`), among others quantities. The value of π_0 can be obtained from a pilot study. If a pilot study is not available, a guess based on a biological system under study could be used. In this case, we recommend using a conservative guess (bigger values for π_0) so that the desired power will still be achieved. The input π_0 can be a vector, in which case separate calculations are performed for each element of the vector. This allows one to assess the changes of power due to the changes in the π_0 estimate(s).

For functions that assume variances follow an Inverse Gamma distribution, it is important to note that if $\frac{1}{\sigma^2} \sim$Gamma(α, β) with mean $\alpha\beta$, version 1.1 of the **ssize.fdr** package uses the parameterization that $\sigma^2 \sim$Inverse Gamma(α, β^{-1}).

Each function outputs the following results: a plot of power vs. sample size, the smallest sample size that achieves the desired power, a table of calculated powers for each sample size, and a table of calculated critical values for each sample size.

Examples

Unless otherwise noted, all data sets analyzed in this section can be downloaded at `http://bioinf.wehi.edu.au/limmaGUI/DataSets.html`.

Using `ssize.oneSamp` and `ssize.oneSampVary`

To illustrate the use of the functions `ssize.oneSamp` and `ssize.oneSampVary`, a data example from Smyth (2004) will be used. In this experiment, zebrafish are used to study early development in vertebrates. Swirl is a point mutation in the BMP2 gene in zebrafish that affects the dorsal/ventral body axis. The goal of this experiment is to identify gene expression levels that differ between zebrafish with the swirl mutation and wild-type zebrafish. This data set (Swirl) includes 8448 gene expression levels from both types of the organism. The experimental design includes two sets of dye-swap pairs. For more details on the microarray technology used and the experimental design of this experiment, see Smyth (2004). The **limma** package performs both normalization and analysis of microarray data sets. See Section 11.1 of Smyth et al. (2008) for complete analysis of the Swirl data set, along with the R code for this analysis.

After normalizing the data, the **limma** package is applied to test if genes are differentially expressed between groups, in this case genotypes (Smyth et al., 2008). This is done by determining if the $\log_2$ ratio of mean gene expressions between swirl mutant zebrafish and the wild type zebrafish are significantly different from zero or not. We then apply the `qvalue` function (in the **qvalue** package) to the vector of p-values obtained from the analysis, and π_0 is estimated to be 0.63.

Now suppose we are interested in performing a similar experiment and want to find an appropriate sample size to use. We want to be able to detect a two-fold change in expression levels between the two genotypes, corresponding to a difference in $\log_2$ expressions of one. We find both up-regulated and down-regulated genes to be interesting, thus our tests will be two-sided. We also want to obtain 80% power while controlling FDR at 5%. We will use the estimated value of π_0 (0.63) from the available data

in addition to other values for π_0 to estimate sample size.

The `ssize.oneSamp` function: To illustrate the use of the `ssize.oneSamp` function, we need to assign a common value for the standard deviations of all genes, so we choose the 90^{th} percentile of the sample standard deviations from the swirl experiment. This value is 0.44. It is important to note that power calculations will be overestimated for genes with higher standard deviation than this. Similarly, power will be conservative for genes with smaller standard deviation than 0.44. To calculate appropriate sample sizes using `ssize.oneSamp`, the following code is used.

```
> os <- ssize.oneSamp(delta=1, sigma=0.44,
    fdr=0.05, power=0.8,
    pi0=c(0.5, 0.63, 0.75, 0.9), maxN=10)
```

This code produces the graph shown in Figure 1

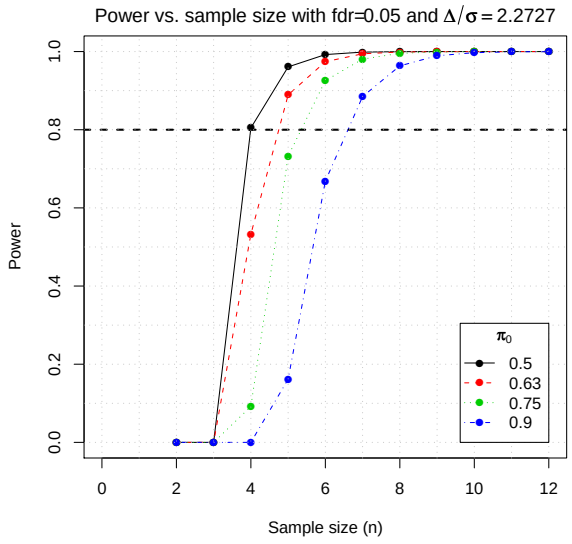


Figure 1: Sample size vs. power for one-sample two-sided t-test with effect size of one, standard deviation of 0.44, and various proportions of non-differentially expressed genes with FDR controlled at 5%.

We see that appropriate samples sizes are 4, 5, 6, and 7 for the above experiment at π_0 values of 0.50, 0.63, 0.75, and 0.90. Notice that although a sample size of five was calculated to be adequate for $\pi_0 = 0.63$, we are performing a dye-swap experiment, so it is better to have an even number of slides. Because of this, we recommend six slides for a dye-swap experiment with two pairs. Sample size information, along with the calculated powers and critical values can be obtained using the following code.

```
> os$ssize
> os$power
> os$crit.vals
```

The `ssize.oneSampVary` function: Now suppose we are interested in representing the standard deviations of the log ratios more precisely. This can be done using a Bayesian approach. Smyth (2004) models the variances of normalized expression levels as

$$\frac{1}{\sigma_g^2} \sim \frac{1}{d_0 s_0^2} \chi^2_{d0} \tag{2}$$

where d_0 and s_0^2 are hyperparameters and can be estimated based on observed residual variances. Performing a simple transformation, we see that (2) is equivalent to

$$\sigma_g^2 \sim \text{Inverse Gamma}\left(\frac{d_0}{2}, \frac{d_0 s_0^2}{2}\right) \tag{3}$$

where the expected value of σ_g^2 is $d_0 s_0^2/(d_0 - 2)$. Using the `lmFit` and `eBayes` functions in the **limma** package, d_0 and s_0^2 can be calculated from any microarray data set. For the Swirl data set, these values are calculated to be $d_0 = 4.17$ and $s_0^2 = 0.051$ (Smyth, 2004). From (3), we see that the variances of the log ratios follow an Inverse Gamma distribution with shape parameter 2.09 and scale parameter 0.106. From this information, we find appropriate sample sizes for a future experiment using the `ssize.oneSampVary` function as follows.

```
> osv <- ssize.oneSampVary(deltaMean=1,
    deltaSE=0, a=2.09, b=0.106,
    fdr=0.05, power=0.8,
    pi0=c(0.5, 0.63, 0.75, 0.9), maxN=15)
```

Figure 2 shows the resulting plot of average power versus sample size for each proportion of non-differentially expressed genes that was input. Note that we are simply interested in finding genes that have a two-fold change in expression levels, so we want the effect size to be held constant at one. This is done by setting the mean of the effect sizes (`deltaMean`) at the desired value with standard deviation (`deltaSE`) equal to zero.

From Figure 2, we see that the appropriate sample sizes are 3, 4, 4, and 5, respectively, to achieve 80% average power with a FDR of 5%. These values are smaller than those in the previous example. In order to obtain appropriate sample size information more directly, along with the calculated powers and critical values, use the following code.

```
> osv$ssize
> osv$power
> osv$crit.vals
```

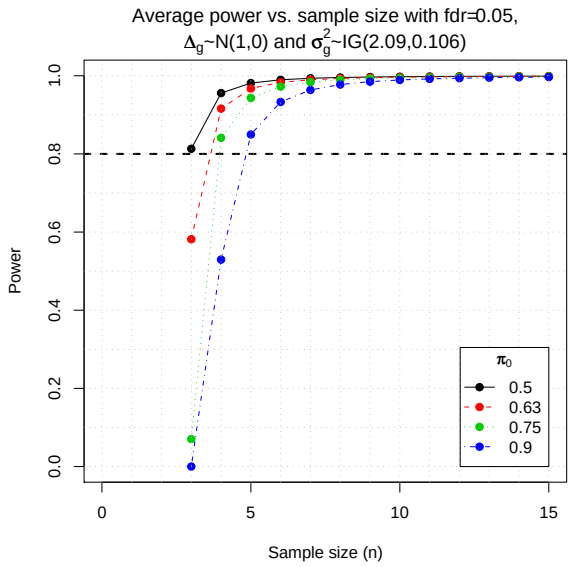


Figure 2: Sample size vs. power for one-sample two-sided t-test with effect size of one and various proportions of non-differentially expressed genes with FDR controlled at 5%. Variances are assumed to follow an Inverse Gamma(2.09, 0.106) distribution.

Using `ssize.twoSamp` and `ssize.twoSampVary`

In order to illustrate the uses of the `ssize.twoSamp` and `ssize.twoSampVary` functions, another data example from Smyth (2004) will be used. The ApoAI gene plays an important role in high density lipoprotein (HDL) metabolism, as mice with this gene knocked out have very low HDL levels. The goal of this study is to determine how the absence of the ApoAI gene affects other genes in the liver. In order to do this, gene expression levels of ApoAI knockout mice and control mice were compared using a common reference design. See Smyth (2004) for a more in-depth description of the study. This is an example of a two sample microarray experiment. For full analysis of this data sets using the **limma** package, see Section 11.4 of Smyth et al. (2008).

Similar to the previous example, the **limma** package is applied to analyze the data. From the resulting p-values, we use the `qvalue` function to estimate the value of π_0 as 0.70.

The `ssize.twoSamp` function: To illustrate the use of the `ssize.twoSamp` function, we can choose a common value to represent the standard deviation for all genes. Similar to the `ssize.oneSamp` example, we can use the 90^{th} percentile of the gene residual standard deviations. In this case, this value is 0.42. Again assume we are interested in genes showing a two-fold change, but this time we are only concerned with up-regulated genes, so we will be performing a one-sided upper tail t-test for each gene. We also want to achieve 80% power while controlling FDR at 5%. The `ssize.twoSamp` function can be used to calculate an appropriate sample size for a simliar experiment with the following code. This code also produces Figure 3.

```
> ts <- ssize.twoSamp(delta=1, sigma=0.42,
    fdr=0.05, power=0.8,
    pi0=c(0.6, 0.7, 0.8, 0.9),
    side="upper", maxN=15)}
```

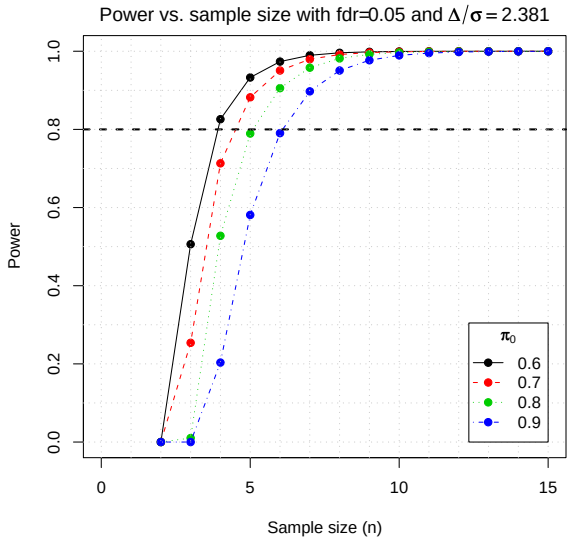


Figure 3: Sample size vs. power for two-sample one-sided upper t-test with effect size of one, standard deviation of 0.42, and various proportions of non-differentially expressed genes with FDR controlled at 5%.

From Figure 3, we see that appropriate sample sizes are 4, 5, 6, and 7 for each genotype group for π_0 values of 0.6, 0.7, 0.8, and 0.9, respectively. Calculated critical values and power along with the sample size estimates can be obtained with the following code.

```
> ts$ssize
> ts$power
> ts$crit.vals
```

The `ssize.twoSampVary` function: As in the `ssize.oneSampVary` example, the **limma** package calculates d_0 to be 3.88 and s_0^2 to be 0.05 as in (2) and (3) using the ApoAI data. From (3), it follows that the variances of the gene expression levels follow an Inverse Gamma distribution with shape parameter 1.94 and scale parameter 0.10. With these values, we can use the `ssize.twoSampVary` function to calculate appropriate sample sizes for experiments for detecting a fold change of one in up-regulated genes. We also want our results to have at least 80% average power with FDR conrolled at 5%.

```
> tsv <- ssize.twoSampVary(deltaMean=1,
    deltaSE=0, a=1.94, b=0.10,
```

```
fdr=0.05, power=0.8,
pi0=c(0.6, 0.7, 0.8, 0.9),
side="upper", maxN=15)
```

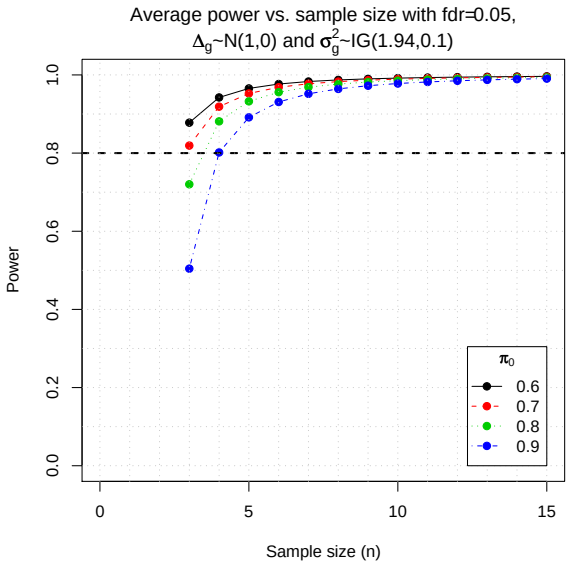


Figure 4: Sample size vs. power for two-sample one-sided upper t-test with effect size of one, gene variances following an Inverse Gamma(1.94, 0.10) distribution, and various proportions of non-differentially expressed genes with FDR controlled at 5%.

Figure (4) shows that appropriate sample sizes have decreased from the previous example (3, 3, 4, and 4 for π_0 values of 0.6, 0.7, 0.8, 0.9). To get this information along with the power and critical value estimates, the following code can be used.

```
> tsv$ssize
> tsv$power
> tsv$crit.vals
```

Using `ssize.F` and `ssize.Fvary`

Lastly, we will use data from an experiment with a 2x2 factorial design to illustrate the uses of the `ssize.F` and `ssize.Fvary` functions. The data in this case are gene expression levels from MCF7 breast cancer cells obtained from Affymetrix HGV5av2 microarrays. The two treatment factors are estrogen (present/absent) and exposure length (10 hours/48 hours), and the goal of this experiment is to identify genes that respond to an estrogen treatment and classify these genes as early or late responders. Data for this experiment can be found in the **estrogen** package available at `http://www.bioconductor.org`. For full analysis of this data set using the **limma** package and more details of the experiment, see Section 11.4 of Smyth et al. (2008).

Because there are two factors with two levels each, there are a total of four treatments, and we define their means of normalized $\log_2$ expression values for a given gene g in the table below. In this table, μ represents the mean of normalized gene expression levels for cells with no estrogen at 10 hours, τ represents the effect of time in the absence of estrogen, α represents the change for cells with estrogen at 10 hours, and γ represents the change for cells with estrogen at 48 hours Smyth et al. (2008).

Treatment	Mean
no estrogen, 10 hours	μ
estrogen, 10 hours	$\mu+\alpha$
no estrogen, 48 hours	$\mu+\tau$
estrogen, 48 hours	$\mu+\tau+\gamma$

One design matrix that corresponds to this experiment and the means in the table above is

$$X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

with parameter vector $\beta = [\mu, \tau, \alpha, \gamma]$. Notice that for this design, there are four parameters to estimate and four slides for each sample, thus the degrees of freedom are $4n-4$. Because we are only interested in the effect of estrogen and how this changes over time, we only care about the parameters α and γ. Thus, we let L be the matrix of the linear contrasts of interest or

$$L = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

so that $L'\beta = [\alpha, \gamma]$.

In order to calculate samples sizes for a similar experiment that will result in tests with 80% power and FDR controlled at 5%, we must have a reasonable estimate for π_0. Using the `lmFit` and `eBayes` functions of the `limma` package in conjunction with `qvalue`, we obtain an estimate value for π_0 of 0.944. Similar to the other examples, we will include this value along with other similar values of π_0 to see how power varies.

The `ssize.F` function: As in all of the previous examples, we find the 90^{th} percentile of the sample residual standard deviations to be 0.29 (or we can find any other percentile that is preferred). Also, let the true parameter vector be $\beta = [12, 1, 1, 0.5]$. Note that the values of the μ and τ parameters do not affect any sample size calculations because we are only interested in α and γ, as represented by $L'\beta$. Thus, the following code will calculate the sample size required to achieve 80% power while controlling the FDR at 5%.

```
> X <- matrix(c(1,0,0,0,1,0,1,0,1,1,0,0,1,1,0,1),
              nrow=4, byrow=TRUE)
> L <- matrix(c(0,0,0,0,1,0,0,1),
              nrow=4, byrow=TRUE)
> B <- c(12, 1, 1, 0.5)
```

```
> dn <- function(n){4*n - 4}
> fs <- ssize.F(X=X, beta=B, L=L, dn=dn,
      sigma=0.29, pi0=c(0.9, 0.944, 0.975, 0.995),
      maxN=12)
```

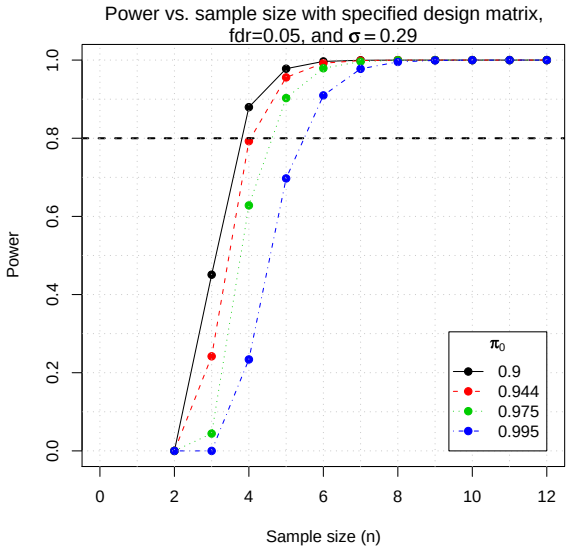


Figure 5: Sample size vs. power for F-test with parameter vector $\beta = [12,1,1,0.5]$, common gene expression standard deviation of 0.29, and various proportions of non-differentially expressed genes with FDR controlled at 5%.

From Figure 5, we see that appropriate sample sizes for this experiment are 4, 5, 5, and 6, for π_0 values of 0.9, 0.944, 0.975, and 0.995, respectively. It is important to note that in this experiment, one sample includes a set of four gene chips (one for each treatment). So a sample size of 4 would require a total of 16 chips, with 4 for each of the four treatment groups.

For other useful information, use the following code.

```
> fs$ssize
> fs$power
> fs$crit.vals
```

The `ssize.Fvary` function: Using the `limma` package, d_0 and s_0^2 are calculated as 4.48 and 0.022, where d_0 and s_0^2 are as in (2). As shown in (3), this corresponds to inverse gamma parameters for σ_g^2 of 2.24 and 0.049. Using the same parameter vector as in the previous example, sample sizes for future experiments can be calculated with the following commands, where we wish to control the FDR at 5% and achieve an average test power of 80%.

```
> fsv <- ssize.Fvary(X=X, beta=B, L=L,
    dn=dn, a=2.24, b=0.049,
    pi0=c(0.9, 0.944, 0.975, 0.995),
    maxN=12)}
```

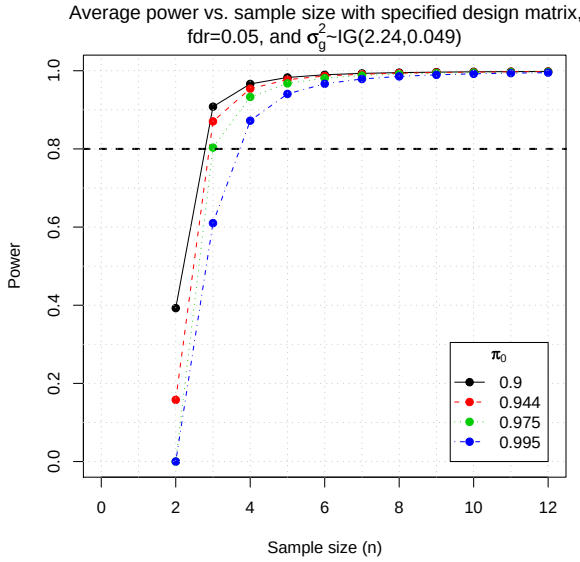


Figure 6: Sample size vs. power for F-test with parameter vector $\beta = [12,1,1,0.5]$, gene expression variances following an Inverse Gamma(2.24,0.049), and various proportions of non-differentially expressed genes with FDR controlled at 5%.

Figure 6 suggests that appropriate sample sizes for this experiment is three for the smallest three π_0 values and four for π_0=0.995. For calculated sample sizes, critical values, and powers, use the following code.

```
> fsv$ssize
> fsv$power
> fsv$crit.vals
```

Modifications

Functions `ssize.oneSampVary`, `ssize.twoSampVary`, and `ssize.Fvary` calculate sample sizes using the assumption that at least one parameter is random. Currently, these assumptions are that effect sizes follow Normal distributions and variances follow Inverse Gamma distributions. If users desire to assume that parameters follow other distributions, the code can be modified accordingly.

Acknowledgement

This material is based upon work partially supported by the National Science Foundation under Grant Number 0714978.

Bibliography

P. Liu and J. T. G. Hwang. Quick calculation for sample size while controlling false discovery rate with

application to microarray analysis. *Bioinformatics*, 23(6):739–746, 2007.

G. K. Smyth. Linear models and empirical bayes methods for assessing differential expression in microarray experiments. *Statistical Applications in Genetics and Molecular Biology*, 3(1):1–4, 2004.

G. K. Smyth, M. Ritchie, N. Thorne, and J. Wettenhall. limma: Linear models for microarray data user's guide. 2008. URL http://bioconductor.org/packages/2.3/bioc/vignettes/limma/inst/doc/usersguide.pdf.

J. D. Storey and R. Tibshirani. Statistical significance for genomewide studies. *Proc. Natl. Acad. Sci. USA*, 100(16):9440–9445, 2003.

Megan Orr
Department of Statistics & Statistical Laboratory
Iowa State University, USA
meganorr@iastate.edu

Peng Liu
Department of Statistics & Statistical Laboratory
Iowa State University, USA
pliu@iastate.edu

Easier Parallel Computing in R with snowfall and sfCluster

by Jochen Knaus, Christine Porzelius, Harald Binder and Guido Schwarzer

Many statistical analysis tasks in areas such as bioinformatics are computationally very intensive, while lots of them rely on embarrassingly parallel computations (Grama et al., 2003). Multiple computers or even multiple processor cores on standard desktop computers, which are widespread nowadays, can easily contribute to faster analyses.

R itself does not allow parallel execution. There are some existing solutions for R to distribute calculations over many computers — a *cluster* — for example **Rmpi**, **rpvm**, **snow**, **nws** or **papply**. However these solutions require the user to setup and manage the cluster on his own and therefore deeper knowledge about cluster computing itself is needed. From our experience this is a barrier for lots of R users, who basically use it as a tool for statistical computing.

Parallel computing has several pitfalls. A single program can easily affect a complete computing infrastructure on maloperation such as allocating too many CPUs or RAM leaving no resources for other users and their processes, or degrading the performance of one or more individual machines. Another problem is the difficulty of keeping track of what is going on in the cluster, which is sometimes hard if the program fails for some reason on a slave.

We developed the management tool sfCluster and the corresponding R package **snowfall**, which are designed to make parallel programming easier and more flexible. sfCluster completely hides the setup and handling of clusters from the user and monitors the execution of all parallel programs for problems affecting machines and the cluster. Together with **snowfall** it allows the use of parallel computing in R without further knowledge of cluster implementation and configuration.

snowfall

The R package **snowfall** is built as an extended abstraction layer above the well established **snow** package by L. Tierney, A. J. Rossini, N. Li and H. Sevcikova (Rossini et al., 2007). Note this is not a technical layer, but an enhancement in useablity. **snowfall** can use all networking types implemented in **snow**, which are socket, MPI, PVM and NetWorkSpaces.

snowfall is also usable on its own (without sfCluster), which makes it handy on single multicore machines, for development or for distribution inside packages.

The design goal was to make parallel computing accessible to R programmers without further general computing knowledge. The Application Programming Interface (API) of **snowfall** is similar to **snow** and indeed **snow** functions can be called directly. So porting of existing **snow** programs is very simple.

The main **snowfall** features are as follows:

- All cluster functions and **snow** wrappers include extended error handling with stopping on error, which makes it easier to find improper behaviour for users without deeper knowledge of clusters.
- In addition to **snow** functions, there are several functions for common tasks in parallel computing. For example functions for loading packages and sources in the cluster and exchanging variables between cluster nodes are present, as well as helpers for saving intermediate results during calculation.
- Cluster settings can be controlled with command line arguments. Users do not have to change their R scripts to switch between sequential or parallel execution, or to change the number of cores or type of clusters used.
- Connector to sfCluster: Used with sfCluster, configuration does not have to be done on initialisation as all values are taken from the user-given sfCluster settings.
- All functions work in sequential execution, too, i.e. without a cluster. This is useful for development and distribution of packages using **snowfall**. Switching between sequential and parallel execution does not require code changes inside the parallel program (and can be changed on initialisation).

Like **snow**, **snowfall** basically uses list functions for parallelisation. Calculations are distributed on slaves for different list elements. This is directly applicable to any data parallel problem, for example bootstrapping or cross-validation can be represented with this approach (other types of parallelisation can also be used, but probably with more effort).

Generally, a cluster constitutes single machines, called *nodes*, which are chosen out of a set of all machines usable as nodes, called the *universe*. The calculation is started on a *master* node, which spawns *worker* R processes (sometimes also called *slaves*). A CPU is a single calculation unit of which modern computers may have more than one.

```
library(snowfall)
# 1. Initialisation of snowfall.
# (if used with sfCluster, just call sfInit())
sfInit(parallel=TRUE, cpus=4, type="SOCK")

# 2. Loading data.
require(mvna)
data(sir.adm)

# 3. Wrapper, which can be parallelised.
wrapper <- function(idx) {
  # Output progress in worker logfile
  cat( "Current index: ", idx, "\n" )
  index <- sample(1:nrow(sir.adm), replace=TRUE)
  temp <- sir.adm[index, ]
  fit <- crr(temp$time, temp$status, temp$pneu)
  return(fit$coef)
}

# 4. Exporting needed data and loading required
#    packages on workers.
sfExport("sir.adm")
sfLibrary(cmprsk)

# 5. Start network random number generator
#    (as "sample" is using random numbers).
sfClusterSetupRNG()

# 6. Distribute calculation
result <- sfLapply(1:1000, wrapper)

# Result is always in list form.
mean(unlist(result))

# 7. Stop snowfall
sfStop()
```

Figure 1: Example bootstrap using snowfall.

For most basic programs, the parallelisation follows the workflow of the example in Figure 1.

1. Initialisation. Call `sfInit` with parameters if not using sfCluster or without parameters if used with sfCluster. Parameters are used to switch between parallel or sequential execution (argument `parallel`, default `FALSE`) and the number of CPUs wanted (argument `cpus` with numerical value, in sequential mode always '1', in parallel mode the default is '2'). Used with sfCluster, these parameters are taken from the sfCluster settings.

2. Load the data and prepare the data needed in the parallel calculations (for example generating data for a simulation study).

3. Wrap parallel code into a wrapper function (function `wrapper` in the example), callable by an R list function.

4. Export objects needed in the parallel calculation (e.g., `sir.adm`) to cluster nodes. It is also necessary to load required packages on all workers. Exporting objects can reduce the total amount of data transmitted. If there are only a few objects needed in the parallel function, you can export them implicitly using additional arguments in the wrapper function and specifying them in the parallel call (e.g. `sfLapply`).

5. Optional step: start a network random number generator. Such random number generators ensure that nodes produce independent sequences of random numbers. The sequences, and hence results relying on random numbers, are reproducible provided that the same number of workers process the same sequence of tasks.

6. Distribute the calculation to the cluster by using a parallel list function (`sfLapply` in the example). This function distributes calls of `wrapper` to workers (which usually means index 1 to index n is called on CPU 1 to CPU n respectively. These calls are then executed in parallel. If the list is longer than the amount of CPUs, index $n + 1$ is scheduled on CPU 1 again [1]). That also means all used data inside the wrapper function must be exported first, to have them existing on any node (see point 2).

7. Stop cluster via `sfStop()` (If used with sfCluster this is not stopping the cluster itself, but allows reinitialisation with `sfInit`).

Probably the most irritating thing in the example is the export of the data frame. On the provided type of cluster computing, the source program runs only on the master first of all. Local variables and objects remain on the master, so workers do not automatically have access to them. All data, functions and packages needed for the parallel calculation have to be transfered to the workers' processes first. *Export* means objects are instantiated on the slaves. Unlike **snow**'s export function, local variables can be exported, too. As a further addition, all variables can be exported or removed from the worker processes.[2]

Basic networking parameters (like execution mode and the number of CPUs on each machine) can be set on the command line, as seen in Figure 2. Arguments provided in `sfInit()` will override the command line arguments. For example, a script that always uses MPI clusters might include `sfInit(type="MPI")`. This mechanism can be

[1] If calls to the wrapper function take different times, as with search problems, or you have computers with different speeds, most likely you will want to use a *load balanced* version, like sfClusterApplyLB, which dynamically re-schedules calls of `wrapper` to CPUs which have finished their previous job.

[2] There are more elaborate ways to integrate data transfer, e.g. NetWorkSpaces, but from our experience, **snowfall**'s exporting functions are enough for most common needs.

```
# Start a socket cluster on local machine using 3 processors
R CMD BATCH myParPrg.R --args --parallel --cpus=3

# Start a socket cluster with 5 cores (3 on localhost, 2 on machine "other")
R --args --parallel --hosts=localhost:3,other:2 < myParPrg.R

# Start using MPI with 5 cores on R interactive shell.
R --args --parallel --type=MPI --cpus=5
```

Figure 2: Examples for snowfall configuration using the command line.

used in R scripts using **snowfall**, as a connector to sfCluster, or as a binding to other workload and management tools.

In all current parallel computing solutions intermediate results are lost if the cluster dies, perhaps due to a shutdown or crash of one of the used machines. **snowfall** offers a function which saves all available parts of results to disc and reloads them on a restored run. Indeed it does not save each finished part, but any number of CPUs parts (for example: Working on a list with 100 segments on a 5 CPU cluster, 20 result steps are saved). This function cannot prevent a loss of results, but can greatly save time on long running clusters. As a side effect this can also be used to realise a kind of "dynamic" resource allocation: just stop and restart with restoring results on a differently sized cluster.

Note that the state of the RNG is not saved in the current version of **snowfall**. Users wishing to use random numbers need to implement customized save and restore functions, or use pre-calculated random numbers.

sfCluster

sfCluster is a Unix commandline tool which is built to handle cluster setup, monitoring and shutdown automatically and therefore hides these tasks from the user. This is done as safely as possible enabling cluster computing even for inexperienced users. Using **snowfall** as the R frontend, users can change resource settings without changing their R program. sfCluster is written in Perl, using only Open Source tools.

On the backend, sfCluster is currently built upon MPI, using the LAM implementation (Burns et al., 1994), which is available on most common Unix distributions.

Basically, a cluster is defined by two resources: CPU and memory. The monitoring of memory usage is very important, as a machine is practically unusable for high performance purposes if it is running out of physical memory and starts to swap memory on the hard disk. sfCluster is able to probe memory usage of a program automatically (by running it in sequential mode for a certain time) or to set the upper bound to a user-given value.

The resource allocation can be widely configured: Even partial usage of a specific machine is possible (for example on a machine with 4 CPUs, which is used for other purposes as well, it is possible to allow only e.g. 2 cores for usage in clusters and sfCluster ensures that no more than 2 CPUs are used by parallel computing on this machine). The restriction of usage can leave calculation resources for other tasks, e.g. if a computer is used for other calculations or perhaps a computing pool for students is used for cluster programs, leaving enough CPU power for the users of those machines.

sfCluster checks the cluster universe to find machines with free resources to start the program (with the desired amount of resources — or less, if the requested resources are not available). These machines, or better, parts of these machines, are built into a new cluster, which belongs to the new program. This is done via LAM sessions, so each program has its own independent LAM cluster.

Optionally sfCluster can keep track of resource usage during runtime and can stop cluster programs if they exceed their given memory usage, machines start to swap, or similar events. On any run, all spawned R worker processes are detected and on shut down it is ensured that all of them are killed, even if LAM itself is not closed/shut down correctly.

The complete workflow of sfCluster is shown in Figure 3.

An additional mechanism for resource administration is offered through user groups, which divide the cluster universe into "subuniverses". This can be used to preserve specific machines for specific users. For example, users can be divided in two groups, one able to use the whole universe, the other only specific machines or a subset of the universe. This feature was introduced, because we interconnect the machine pools from two institutes to one universe, where some scientists can use all machines, and some only the machines from their institute.

sfCluster features three main parallel execution modes. All of these setup and start a cluster for the program as described, run R and once the program has finished, shutdown the cluster. The batch- and monitoring mode shuts down the cluster on interruption from the user (using keys `Ctrl-C` on most Unix systems).

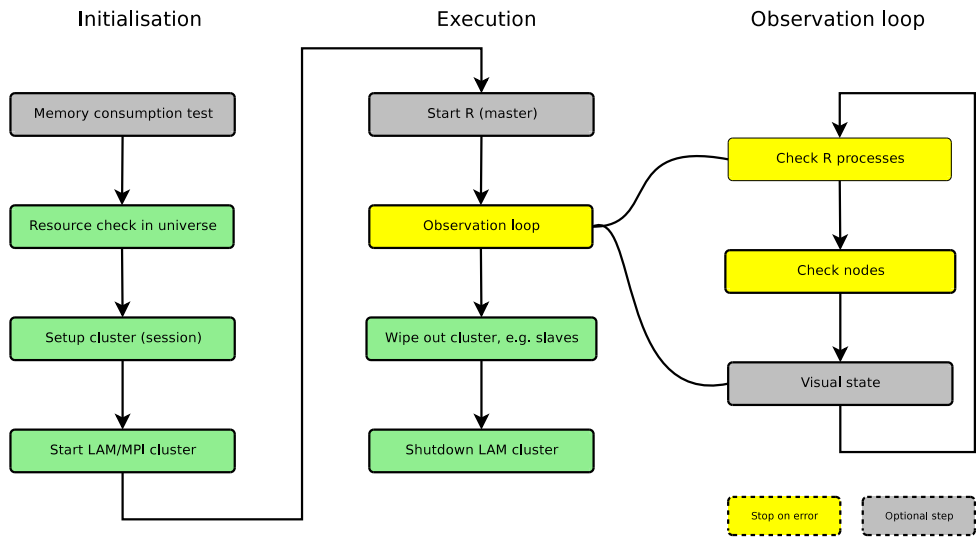


Figure 3: Workflow of sfCluster.

1. Parallel batchmode: Parallel counterpart to `R CMD BATCH`. Called using `sfCluster -b` or by the optionally installed `R CMD par`.

2. Interactive R shell (`sfCluster -i` or `R CMD parint`).

3. Monitoring mode (`sfCluster -m` or `R CMD parmon`): Behaves similarly to batchmode, but features a process monitor for workers, access to worker logfiles, system output and debugging and runtime messages. The visualisation in the terminal is done using Ncurses. For an example output see Figure 4: A cluster with 9 worker processes (marked `SL`) and one master process (marked `MA`) are running on five machines. Each machine has a tab with logfiles marked by the name of the node, e.g. `knecht4`. For each process, it's process identification number (PID), the node it is running on, its memory usage and state (like running, sleeping, dead etc.) is shown at the top. The `System` tab contains system messages from sfCluster; the R output on the master is shown in tab `R-Master`.

Besides the parallel execution modes, the sequential mode can be chosen, which works in all cases, even without an installed or running cluster environment, i.e. also on Windows systems or as part of a package build on **snowfall**. On execution in sequential mode, **snowfall** is forced to run in non-parallel mode.

Besides choosing the amount of required CPUs, users can specify several options on starting sfCluster. These contain for example the R version, the nice level of the slaves, activating the sending of emails after finish or failure and many more.

Administration

sfCluster includes features to get information about current running clusters and free resources on the cluster universe (see Figure 5 for an example). This can help users to get an overview of what is currently happening on the cluster machines and of the resources available for starting their own programs. Also detailed information about clusters, with all process PIDs, memory use and runtime of single processes can be printed, which is useful for administrators to determine which R process belongs to which cluster.

All these administration options are directly usable by (administrative) root account, as well, so administrators can safely kill clusters without wiping out all the slave processes manually.

The configuration of sfCluster itself is done via common Unix-style configuration files. Configuration includes system variables (like definition of triggers for stopping events on observation), the user groups, R versions and of course the cluster universe with resource limits.

The installation is available through tar.gz or as a Debian package; both are instantly usable for a single (probably multicore) machine and only need to be configured for "real" clusters. In normal cases the default installation should work out of the box. It needs some minor tweaking if R is not installed in the default way, e.g. if multiple R installations are available on the machines. All needed CRAN and CPAN packages are also installable through the installer.

Future additions will be a port to OpenMPI, integration to common batch and resource management systems (e.g. the Sun Grid Engine or slurp) and basic reports about cluster behaviour.

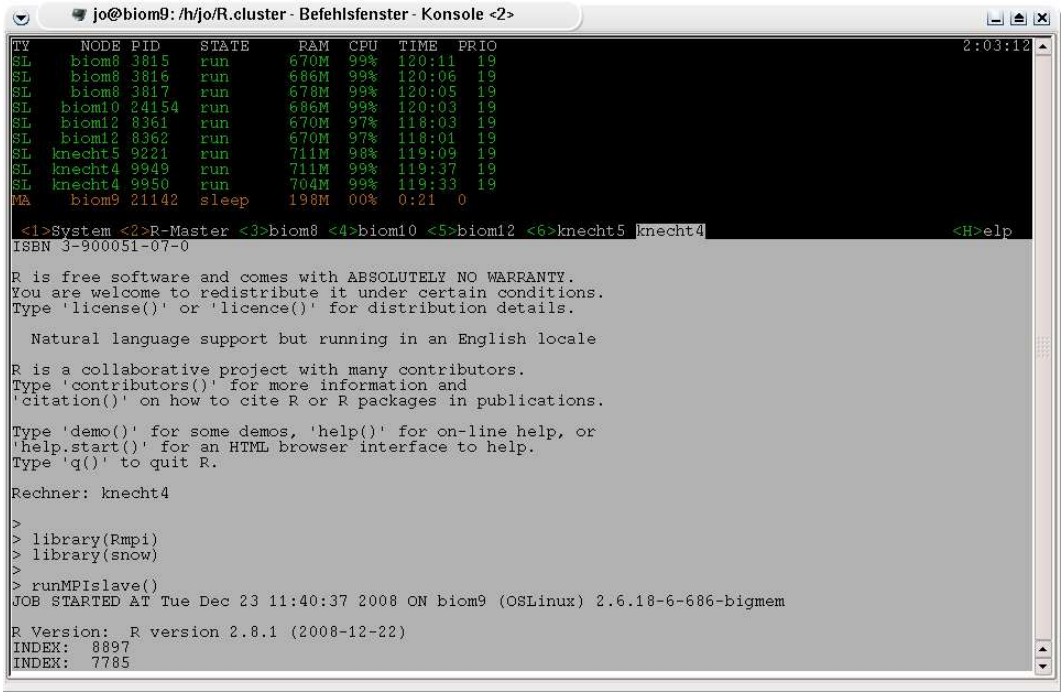


Figure 4: Monitoring mode (9 slaves on five machines and master).

Summary

Although many well-working solutions for parallel computing in R are available, they have their downsides on forcing the user to manage the underlying clusters manually. sfCluster/**snowfall** solves this problem in a very flexible and comfortable way, enabling even inexperienced computer users to benefit from parallel programming (without having to learn cluster management, usage and falling into pitfalls affecting other people's processes). **snowfall** makes sure the resulting R programs are runable everywhere, even without a cluster.

The combination **snowfall**/sfCluster has been used daily in our institute for several months and has evolved with user's demands and wishes. There are packages that have been built integrating **snowfall** with optionally usable parallelisation techniques (e.g. the package **peperr**).

The software and further information are available at http://www.imbi.uni-freiburg.de/parallel.

Acknowledgments

This research was supported by the Deutsche Forschungsgemeinschaft (German Research Foundation) with FOR 534.

Thanks to Arthur Allignol for his little bootstrapping example.

Bibliography

A. Grama, G. Karypis, V. Kumar, A. Gupta. *Introduction to Parallel Computing*. Pearson Eduction, second edition, 2003.

G. Burns, R. Daoud, and J. Vaigl. LAM: An Open Cluster Environment for MPI. Technical report, 1994. http://www.lam-mpi.org/download/files/lam-papers.tar.gz.

A. Rossini, L. Tierney, and N. Li. Simple parallel statistical computing in R. *Journal of Computational and Graphical Statistics*, 16(2):399–420, 2007.

Jochen Knaus, Christine Porzelius, Harald Binder and Guido Schwarzer
Department of Medical Biometry and Statistics
University of Freiburg
Germany
jo@imbi.uni-freiburg.de

```
jo@biom9:~$ sfCluster -o
SESSION          | STATE |  M | MASTER       #N   RUNTIME R-FILE / R-OUT
-----------------+-------+----+-------------------------------------------------
LrtpdV7T_R-2.8.1 | run   | MO | biom9.imbi    9   2:46:51 coxBst081223.R / coxBst081223.Rout
baYwQ0GB_R-2.5.1 | run   | IN | biom9.imbi    2   0:00:18 -undef- / -undef-

jo@biom9:~$ sfCluster -o --all
SESSION          | STATE | USR |  M | MASTER       #N   RUNTIME R-FILE / R-OUT
-----------------+-------+-----+----+-----------------------------------------------
LrtpdV7T_R-2.8.1 | run   | jo  | MO | biom9.imbi    9   3:16:09 coxBst081223.R / coxBst081223.Rout
jlXUhxtP_R-2.5.1 | run   | jo  | IN | biom9.imbi    2   0:00:22 -undef- / -undef-
bSpNLNhd_R-2.7.2 | run   | cp  | BA | biom9.imbi    8   0:32:57 getPoints11.R / getPoints11.Rout
NPS5QHkK_R-2.7.2 | run   | cp  | MO | biom9.imbi   10   3:50:42 box2.R / box2.Rout

jo@biom9:~$ sfCluster --universe --mem=1G
Assumed memuse: 1024M (use '--mem' to change).

Node                           | Max-Load | CPUs | RAM    | Free-Load | Free-RAM | FREE-TOTAL
-------------------------------+----------+------+--------+-----------+----------+------------
biom8.imbi.uni-freiburg.de     |        8 |    8 |  15.9G |     0     |     9.3G |     0
biom9.imbi.uni-freiburg.de     |        8 |    8 |  15.9G |     0     |    12.6G |     0
biom10.imbi.uni-freiburg.de    |        8 |    8 |  15.9G |     0     |    14.0G |     0
biom12.imbi.uni-freiburg.de    |        2 |    4 |   7.9G |     0     |     5.8G |     0
knecht5.fdm.uni-freiburg.de    |        8 |    8 |  15.7G |     1     |     1.2G |     1
knecht4.fdm.uni-freiburg.de    |        8 |    8 |  15.7G |     1     |     4.3G |     1
knecht3.fdm.uni-freiburg.de    |        5 |    8 |  15.7G |     3     |    11.1G |     3
biom6.imbi.uni-freiburg.de     | no-sched |    4 |   7.9G |     -     |        - |     -
biom7.imbi.uni-freiburg.de     |        2 |    4 |   7.9G |     1     |     2.1G |     1

Potential usable CPUs: 6
```

Figure 5: Overview of running clusters from the user (first call) and from all users (second call). The session column states a unique identifier as well as the used R version. "state" describes whether the cluster is currently running or dead. "USR" contains the user who started the cluster. Column "M" includes the sfCluster running mode: `MO/BA/IN` for monitoring, batch and interative. Column "#N" contains the amount of CPUs used in this cluster. The third call gives an overview of the current usage of the whole universe, with free calculation time ("Free-Load") and useable memory ("Free-RAM").

PMML: An Open Standard for Sharing Models

by Alex Guazzelli, Michael Zeller, Wen-Ching Lin and Graham Williams

Introduction

The **PMML** package exports a variety of predictive and descriptive models from R to the Predictive Model Markup Language (Data Mining Group, 2008). PMML is an XML-based language and has become the de-facto standard to represent not only predictive and descriptive models, but also data pre- and post-processing. In so doing, it allows for the interchange of models among different tools and environments, mostly avoiding proprietary issues and incompatibilities.

The **PMML** package itself (Williams et al., 2009) was conceived at first as part of Togaware's data mining toolkit Rattle, the R Analytical Tool To Learn Easily (Williams, 2009). Although it can easily be accessed through Rattle's GUI, it has been separated from Rattle so that it can also be accessed directly in R.

In the next section, we describe PMML and its overall structure. This is followed by a description of the functionality supported by the **PMML** package and how this can be used in R. We then discuss the importance of working with a valid PMML file and finish by highlighting some of the debate surrounding the adoption of PMML by the data mining community at large.

A PMML primer

Developed by the Data Mining Group, an independent, vendor-led committee (`http://www.dmg.org`), PMML provides an open standard for representing data mining models. In this way, models can easily be shared between different applications, avoiding proprietary issues and incompatibilities. Not only can PMML represent a wide range of statistical techniques, but it can also be used to represent their input data as well as the data transformations necessary to transform these into meaningful features.

PMML has established itself as the lingua franca for the sharing of predictive analytics solutions between applications. This enables data mining scientists to use different statistical packages, including R, to build, visualize, and execute their solutions.

PMML is an XML-based language. Its current 3.2 version was released in 2007. Version 4.0, the next version, which is to be released in 2009, will expand the list of available techniques covered by the standard. For example, it will offer support for time series, as well as different ways to explain models, including statistical and graphical measures. Version 4.0 also expands on existing PMML elements and their capabilities to further represent the diverse world of predictive analytics.

PMML Structure

PMML follows a very intuitive structure to describe a data mining model. As depicted in Figure 1, it is composed of many elements which encapsulate different functionality as it relates to the input data, model, and outputs.

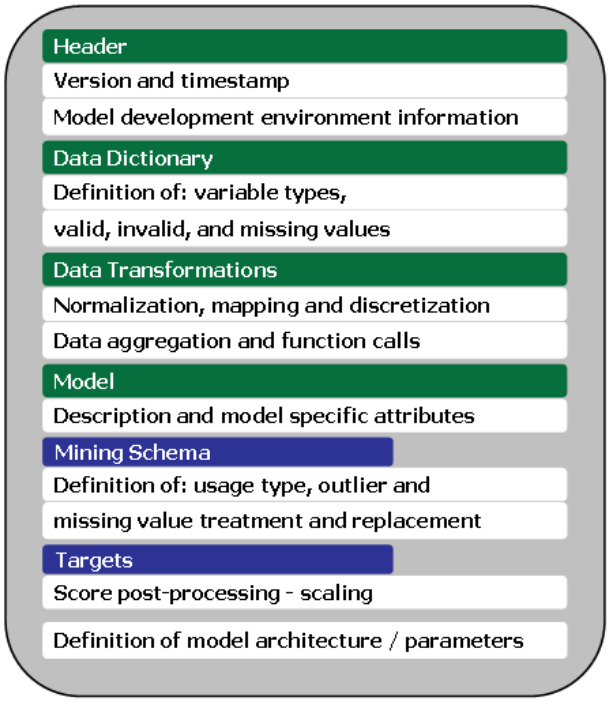


Figure 1: PMML overall structure described sequentially from top to bottom.

Sequentially, PMML can be described by the following elements:

Header

The header element contains general information about the PMML document, such as copyright information for the model, its description, and information about the application used to generate the model such as name and version. It also contains an attribute for a timestamp which can be used to specify the date of model creation.

Data dictionary

The data dictionary element contains definitions for all the possible fields used by the model. It is in the data dictionary that a field is defined as continuous, categorical, or ordinal. Depending on this definition, the appropriate value ranges are then defined as well as the data type (such as string or double). The data dictionary is also used for describing the list of valid, invalid, and missing values relating to every single input data.

Data transformations

Transformations allow for the mapping of user data into a more desirable form to be used by the mining model. PMML defines several kinds of data transformation:

- *Normalization:* Maps values to numbers, the input can be continuous or discrete.
- *Discretization:* Maps continuous values to discrete values.
- *Value mapping:* Maps discrete values to discrete values.
- *Functions:* Derive a value by applying a function to one or more parameters.
- *Aggregation:* Summarizes or collects groups of values.

The ability to represent data transformations (as well as outlier and missing value treatment methods) in conjunction with the parameters that define the models themselves is a key concept of PMML.

Model

The model element contains the definition of the data mining model. Models usually have a model name, function name (classification or regression) and technique-specific attributes.

The model representation begins with a mining schema and then continues with the actual representation of the model:

- *Mining Schema:* The mining schema (which is embedded in the model element) lists all fields used in the model. This can be a subset of the fields defined in the data dictionary. It contains specific information about each field, such as name and usage type. Usage type defines the way a field is to be used in the model. Typical values are: active, predicted, and supplementary. Predicted fields are those whose values are predicted by the model. It is also in the mining schema that special values are treated. These involve:
  - *Outlier Treatment:* Defines the outlier treatment to be used. In PMML, outliers can be treated as missing values, as extreme values (based on the definition of high and low values for a particular field), or as is.
  - *Missing Value Replacement Policy:* If this attribute is specified then a missing value is automatically replaced by the given values.
  - *Missing Value Treatment:* Indicates how the missing value replacement was derived (e.g. as value, mean or median).
- *Targets:* The targets element allows for the scaling of predicted variables.
- *Model Specifics:* Once we have the data schema in place we can specify the details of the actual model.

  A multi-layered feed-forward neural network, for example, is represented in PMML by its activation function and number of layers, followed by the description of all the network components, such as connectivity and connection weights.

  A decision tree is represented in PMML, recursively, by the nodes in the tree. For each node, a test on one variable is recorded and the sub-nodes then correspond to the result of the test. Each sub-node contains another test, or the final decision.

  Besides neural networks and decision trees, PMML allows for the representation of many other data mining models, including linear regression, generalised linear models, random forests and other ensembles, association rules, cluster models, näive Bayes models, support vector machines, and more.

PMML also offers many other elements such as built-in functions, statistics, model composition and verification, etc.

Exporting PMML from R

The **PMML** package in R provides a generic `pmml` function to generate PMML 3.2 for an object. Using an S3 generic function, the appropriate method for the class of the supplied object is dispatched.

An example

A simple example illustrates the steps involved in generating PMML. We will build a decision tree, using **rpart**, to illustrate. With a standard installation of R with the **PMML** package installed, the following should be easily repeatable.

First, we load the appropriate packages and dataset. We will use the well known Iris dataset that records sepal and petal characteristics for different varieties of iris.

```
> library(pmml)
> library(rpart)
> data(iris)
```

We can build a simple decision tree to classify examples into iris varieties. We see the resulting structure of the tree below. The root node test is on the variable *Petal.Length* against a value of 2.45. The "left" branch tests for *Petal.Length* < 2.45 and delivers a decision of *setosa*. The "right" branch splits on a further variable, *Petal.Width*, to distinguish between *versicolor* and *virginica*.

```
> my.rpart <- rpart(Species ~ ., data=iris)
> my.rpart

n= 150

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 150 100 setosa ...
  2) Petal.Length< 2.45 50 0 setosa (0.33 ...
  3) Petal.Length>=2.45 100 50 versicolor ...
    6) Petal.Width< 1.75 54 5 versicolor ...
    7) Petal.Width>=1.75 46 1 virginica ...
```

To convert this model to PMML we simply call the `pmml` function. Below we see some of the output.

The exported PMML begins with a header that contains information about the model. As indicated above, this will contain general information about the model, including copyright, the application used to build the model, and a timestamp. Other information might also be included.

```
> pmml(my.rpart)

<PMML version="3.2" ...
 <Header
  copyright="Copyright (c) 2009 Togaware"
  description="RPart Decision Tree">
  <Extension name="timestamp"
    value="2009-02-15 06:51:50"
    extender="Rattle"/>
  <Extension name="description"
    value="iris tree"
    extender="Rattle"/>
  <Application name="Rattle/PMML"
    version="1.2.7"/>
 </Header>
```

Next, the data dictionary records information about the data fields from which the model was built. Here we see the definition of a categoric and a numeric data field.

```
 <DataDictionary numberOfFields="5">
  <DataField name="Species" ...
   <Value value="setosa"/>
   <Value value="versicolor"/>
   <Value value="virginica"/>
  <DataField name="Sepal.Length"
    optype="continuous"
    dataType="double"/>
  </DataField>
 ...
```

The header and the data dictionary are common to all PMML, irrespective of the model.

Next we record the details of the model itself. In this case we have a tree model, and so a 'TreeModel' element is defined. Within the tree model we begin with the mining schema.

```
 <TreeModel modelName="RPart_Model"
            functionName="classification"
            algorithmName="rpart"
            ...>
  <MiningSchema>
   <MiningField name="Species"
                usageType="predicted"/>
   <MiningField name="Sepal.Length"
                usageType="active"/>
 ...
```

This is followed by the actual nodes of the tree. We can see that the first test involves a test on the *Petal.Length*. Once again, it is testing the value against 2.45.

```
  <Node id="1" score="setosa"
        recordCount="150" defaultChild="3">
   <True/>
   <ScoreDistribution value="setosa"
     recordCount="50" confidence="0.33"/>
   <ScoreDistribution value="versicolor"
     recordCount="50" confidence="0.33"/>
   <ScoreDistribution value="virginica"
     recordCount="50" confidence="0.33"/>
   <Node id="2" score="setosa"
         recordCount="50">
    <CompoundPredicate
       booleanOperator="surrogate">
     <SimplePredicate field="Petal.Length"
         operator="lessThan" value="2.45"/>
...
  </Node>
 </TreeModel>
</PMML>
```

We also note that more information is captured here than displayed with R's print method for the **rpart** object. The **rpart** object in fact includes information about surrogate splits which is also captured in the PMML representation.

Usually, we will want to save the PMML to a file, and this can easily be done using the `saveXML` function from the **XML** package (Temple Lang , 2009).

```
> saveXML(pmml(my.rpart),
+         file="my_rpart.xml")
```

Supported Models

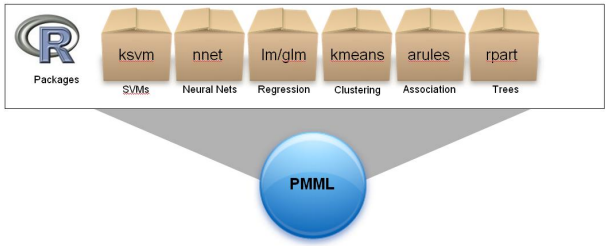


Figure 2: PMML Export functionality is available for several predictive algorithms in R.

Although coverage is constantly being expanded, PMML exporter functionality is currently available for the following data mining algorithms:

1. Support Vector Machines — **kernlab** (Karatzoglou et al., 2008): PMML export for SVMs implementing multi-class and binary classification as well as regression for objects of class `ksvm` (package **kernlab**). It also exports transformations for the input variables by following the scaling scheme used by **ksvm** for numerical variables, as well as transformations to create dummy variables for categorical variables.

   For multi-class classification, **ksvm** uses the *one-against-one* approach, in which the appropriate class is found by a voting scheme. Given that PMML 3.2 does not support this approach, the export functionality comes with an extension piece to handle such cases. The next version of PMML to be released in early 2009 will be equipped to handle different classification methods for SVMs, including one-against-one.

   The example below shows how to train a support vector machine to perform binary classification using the audit dataset — see Williams (2009).

   ```
   > library(kernlab)
   > audit <- read.csv(file(
   +"http://rattle.togaware.com/audit.csv")
   + )
   > myksvm <- ksvm(as.factor(Adjusted) ~ .,
   +             data=audit[,c(2:10,13)],
   +             kernel = "rbfdot",
   +             prob.model=TRUE)
   > pmml(myksvm, data=audit)
   ```

   Note that the **PMML** package is being invoked in the example above with two parameters: 1) the `ksvm` object which contains the model representation; and 2) the data object used to train the model. The **PMML** package uses the data object to be able to retrieve the values for categorical variables which are used internally by `ksvm` to create dummy variables, but are not part of the resulting `ksvm` object.

2. Neural Networks — **nnet**: PMML export for neural networks implementing multi-class or binary classification as well as regression models built with the **nnet** package, available through the **VR** bundle (Venables and Ripley, 2002). Some details that are worth mentioning are:

   - Scaling of input variables: Since **nnet** does not automatically implement scaling of numerical inputs, it needs to be added to the generated PMML file by hand if one is planning to use the model to compute scores/results from raw, unscaled data.
   - The PMML exporter uses transformations to create dummy variables for categorical inputs. These are expressed in the 'NeuralInputs' element of the resulting PMML file.
   - PMML 3.2 does not support the censored variant of softmax.
   - Given that **nnet** uses a single output node to represent binary classification, the resulting PMML file contains a discretizer with a threshold set to 0.5.

3. Classification and Regression Trees — **rpart** (Therneau and Atkinson. R port by B. Ripley, 2008): PMML export functionality for decision trees built with the **rpart** package is able to export classification as well as regression trees. It also exports surrogate predicate information and missing value strategy (default child strategy).

4. Regression Models — `lm` and `glm` from **stats**: PMML export for linear regression models for objects of class `"lm"` and binary logistic regression models for objects of class `"glm"` built with the binomial family. Note that this function currently does not support multinomial logistic regression models or any other regression models built using the **VGAM** package.

5. Clustering Models — `hclust` and `kmeans` from **stats**: PMML export functionality for clustering models for objects of class `"hclust"` and `"kmeans"`.

6. Association Rules — **arules** (Hahsler et al., 2008): PMML export for association rules built with the **arules** package.

7. Random Forest (and randomSurvivalForest) — **randomForest** (Breiman and Cutler. R port by A. Liaw and M. Wiener, 2009) and **randomSurvivalForest** (Ishwaran and Kogalur , 2009): PMML export of a randomSurvivalForest `"rsf"` object. This function gives the user the ability to export PMML containing the geometry of a forest.

PMML validation

Since PMML is an XML-based standard, the specification comes in the form of an XML Schema. Zementis (`http://www.zementis.com`) has built a tool that can be used to validate any PMML file against the current and previous PMML schemas. The tool can also be used to convert older versions of PMML (2.1, 3.0, 3.1) to version 3.2 (the current version). The PMML Converter will validate any PMML file and is currently able to convert the following modeling elements:

1. Association Rules
2. Neural Networks
3. Decision Trees
4. Regression Models
5. Support Vector Machines
6. Cluster Models

The PMML converter is free to use and can be accessed directly from the Zementis website or installed as a gadget in an iGoogle console.

It is very important to validate a PMML file. Many vendors claim to support PMML but such support is often not very complete. Also, exported files do not always conform to the PMML specification. Therefore, interoperability between tools can be a challenge at times.

More importantly, strict adherence to the schema is necessary for the standard to flourish. If this is not enforced, PMML becomes more of a blueprint than a standard, which then defeats its purpose as an open standard supporting interoperability. By validating a PMML file against the schema, one can make sure that it can be moved around successfully. The PMML Converter is able to pinpoint schema violations, empowering users and giving commercial tools, as well as the **PMML** package available for R, an easy way to validate their export functionality. The example below shows part of the data dictionary element for the iris classification tree discussed previously. This time, however, the PMML is missing the required dataType attribute for variable *Sepal.Length*. The PMML Converter flags the problem by embedding an XML comment into the PMML file itself.

```
<DataDictionary>
  <!--PMML Validation Error:
  Expected attribute: dataType in
  element DataField -->
  <DataField name="Sepal.Length"
    optype="continuous">
  </DataField>
  <DataField name="Sepal.Width"
    dataType="double"
    optype="continuous">
</DataField>
...
```

Discussion

Although PMML is an open standard for representing predictive models, the lack of awareness has made its adoption slow. Its usefulness was also limited because until recently predictive models were in general built and deployed using the same data mining tool. But this is now quickly changing with models built in R, for example, now able to be deployed in other model engines, including data warehouses that support PMML.

For those interested in joining an on-going discussion on PMML, we have created a PMML discussion group under the AnalyticBridge community (`http://www.analyticbridge.com/group/pmml`). In one of the discussion forums, the issue of the lack of support for the export of data transformations into PMML is discussed. Unless the model is built directly from raw data (usually not the case), the PMML file that most tools will export is incomplete. This also extends to R. As we saw in the previous section, the extent to which data transformations are exported to PMML in R depends on the amount of pre-processing carried out by the package/class used to build the model. However, if any data massaging is done previous to model building, this needs to be manually converted to PMML if one wants that to be part of the resulting file. PMML offers coverage for many commonly used data transformations, including mapping and normalization as well as several built-in functions for string, date and time manipulation. Built-in functions also offer several mathematical operations as depicted in the PMML example below, which implements: *maximum(round(inputVar/1.3))*.

```
<Apply function="max">
  <Apply function="round">
    <Apply function="/">
      <FieldRef field="inputVar"/>
      <Constant>1.3</Constant>
    </Apply>
  </Apply>
</Apply>
```

Integrated tools that export PMML should allow not only for the exporting of models, but also data transformations. One question we pose in our discussion forum is how to implement this in R.

More recently, KNIME has implemented extensive support for PMML. KNIME is an open-source framework that allows users to visually create data flows (http://www.knime.org). It implements plugins that allow R-scripts to be run inside its Eclipse-based interface. KNIME can import and export PMML for some of its predictive modules. In doing so, it allows for models built in R to be loaded in KNIME for data flow visualization. Given that it exports PMML as well, it could potentially be used to integrate data pre- and post-processing into a single PMML file which would then contain the entire data and model processing steps.

PMML offers a way for models to be exported out of R and deployed in a production environment, quickly and effectively. Along these lines, an interesting development we are involved in is the ADAPA scoring engine (Guazzelli et al., 2009), produced by Zementis. This PMML consumer is able to upload PMML models over the Internet and then execute/score them with any size dataset in batch-mode or real-time. This is implemented as a service through the Amazon Compute Cloud. Thus, models developed in R can be put to work in matter of minutes, via PMML, and accessed in real-time through web-service calls from anywhere in the world while leveraging a highly scalable and cost-effective cloud computing infrastructure.

Bibliography

L. Breiman and A. Cutler. R port by A. Liaw and M. Wiener. *randomForest: Breiman and Cutler's Random Forests for Classification and Regression*, 2009. URL http://cran.r-project.org/package=randomForest. R package version 4.5-30.

Data Mining Group. PMML version 3.2. WWW, 2008. URL http://www.dmg.org/pmml-v3-2.html.

A. Guazzelli, K. Stathatos, and M. Zeller. Efficient deployment of predictive analytics through open standards and cloud computing. *To appear in ACM SIGKDD Explorations*, June 2009. URL http://www.sigkdd.org/explorations.

M. Hahsler, C. Buchta, B. Gruen, and K. Hornik. *arules: Mining Association Rules and Frequent Itemsets*, 2008. URL http://cran.r-project.org/package=arules. R package version 0.6-8.

H. Ishwaran and U. Kogalur. *randomSurvivalForest: Ishwaran and Kogalur's Random Surival Forest*, 2009. URL http://cran.r-project.org/package=randomSurvivalForest. R package version 3.5.1.

A. Karatzoglou, A. Smola, and K. Hornik. *The kernlab package*, 2008. URL http://cran.R-project.org/package=kernlab. R package version 0.9-8.

D. Temple Lang. *XML: Tools for parsing and generating XML within R and S-Plus*, 2009. URL http://cran.R-project.org/package=xml. R package version 2.3-0.

T. M. Therneau and B. Atkinson. R port by B. Ripley. *rpart: Recursive Partitioning*, 2008. URL http://cran.r-project.org/package=rpart. R package version 3.1-42.

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Statistics and Computing. Springer, New York, 4th edition, 2002. URL http://cran.r-project.org/package=VR. R package version 7.2-45.

G. Williams. *rattle: A graphical user interface for data mining in R*, 2009. URL http://cran.r-project.org/package=rattle. R package version 2.4.31.

G. Williams, M. Hahsler, A. Guazzelli, M. Zeller, W. Lin, H. Ishwaran, U. B. Kogalur, and R. Guha. *pmml: Generate PMML for various models*, 2009. URL http://cran.r-project.org/package=pmml. R package version 1.2.7.

Alex Guazzelli, Michael Zeller, Wen-Ching Lin
Zementis Inc
info@zementis.com

Graham Williams
Togaware Pty Ltd
Graham.Williams@togaware.com

News and Notes

Forthcoming Events: Conference on Quantitative Social Science Research Using R

by H. D. Vinod, Fordham University, Bronx, NY 10458

June 18-19 (Thursday-Friday), 2009, Fordham University, 113 West 60th Street, New York, NY. (next door to the Lincoln Center for Performing Arts).

Conference website:
http://www.cis.fordham.edu/QR2009

The conference offers opportunity to enthusiastic users of R to discuss important policy and research problems in social sciences, to learn, meet and mingle.

- Speakers include authors of books and /or packages for R, published researchers and editors of important journals in Statistics and social sciences: Andrew Gelman (Columbia), Kosuke Imai (Princeton), Roger Koenker (Illinois), Keith A. Markus (CUNY), Bruce D. McCullough (Drexel), H. D. Vinod (Fordham), Achim Zeileis (Vienna).
- Conference proceedings book will be published by Springer (publisher of Use R! series) in 2009.
- Opportunity to present Replication / extension of published papers using R, (poster session).
- Opportunity to present new useful R functions (including teaching material) (poster session).
- A tutorial cum refresher session teaching R

Forthcoming Events: DSC 2009

As mentioned in the last issue of *R News*, the sixth international workshop on Directions in Statistical Computing (DSC 2009) will take place at the Center for Health and Society, University of Copenhagen, Denmark, 13th–14th of July 2009. For details, see http://www.R-project.org/dsc-2009.

- The paper selection process is now complete and the programme committee have approximately 25 presentations to assemble into a Final Scientific Programme.
- Professor Tue Tjur from the Copenhagen Business School has been invited to give an opening lecture on statistical computing in the Old Days. (This was the teacher who introduced me to GENSTAT 30 years ago.)
- The Social Programme includes an opening mixer at the conference venue on Sunday evening and a Conference dinner on Monday at Furesø Marina. (http://marina-en.dk)
- Early registration closed on May 15. Regular registration is possible until July 5.
- The registration is now handled by a professional congress bureau. In particular, this means that payment by credit card has become possible.

On behalf of the Organizing Committee,
Peter Dalgaard

Forthcoming Events: useR! 2009

The fifth international R user conference 'useR! 2009' will take place at Agrocampus, Rennes, France, July 8-10, 2009.

This fifth world-meeting of the R user community will focus on

- R as the 'lingua franca' of data analysis and statistical computing,
- providing a platform for R users to discuss and exchange ideas how R can be used to do statistical computations, data analysis, visualization and exciting applications in various fields,
- giving an overview of the new features of the rapidly evolving R project.

The program comprises keynote lectures, user-contributed sessions and pre-conference tutorials. The schedules are available on: http://www.agrocampus-ouest.fr/math/useR-2009/program.html

Keynote Lectures

R has become the standard computing engine in more and more disciplines, both in academia and the business world. How R is used in different areas will be presented in keynote lectures addressing hot topics including

- Random forests (Adele Cutler)
- What we wish people knew more about when working with R? (Peter Dalgaard)
- Fast Sparse Regression and Classification (Jerome H. Friedman)
- Dynamic transitions between multivariate methods (Michael Greenacre)
- Regularization Paths for Generalized Linear Models via Coordinate Descent (Trevor Hastie)
- Whole Data Approaches to Large-Scale Multiple Hypothesis Testing (John Storey)
- Statistical Graphics! Who needs Visual Analytics? (Martin Theus)

User-contributed Sessions

The sessions will be a platform to bring together R users, contributers, package maintainers and developers. People from different fields will show us how they solve problems with R in fascinating applications. The scientific program is organized by members of the program committee, including Douglas Bates, Vincent Carey, Pierre-Andre Cornillon, Jan De Leeuw, Ramon Diaz-Uriarte, Marco A. R. Ferreira, Nicolas Hengartner, Torsten Hothorn, François Husson, Friedrich Leisch, Thomas Petzoldt, Ching-Fan Sheu, Ron Wehrens and Achim Zeileis and will cover topics such as

- Applied Statistics & Biostatistics
- Bayesian Statistics
- Bioinformatics
- Chemometrics and Computational Physics
- Data Mining
- Econometrics & Finance
- Environmetrics & Ecological Modeling
- High Performance Computing
- Machine Learning
- Marketing & Business Analytics
- Psychometrics
- Robust Statistics
- Sensometrics
- Spatial Statistics
- Statistics in the Social and Political Sciences
- Teaching
- Visualization & Graphics
- and many more.

Pre-conference Tutorials

Before the start of the official program, half-day tutorials will be offered on Tuesday, July 7th, including

- Douglas Bates: Mixed models in R using the lme4 package
- Dirk Eddelbuettel: Introduction to High-Performance Computing with R
- Frank E Harrell Jr: Regression Modeling Stategies using the R Design Package
- Sebastien Lê, Jérôme Pagès & Marine Cadoret: From multivariate to multiway data analysis...an overview
- Thomas Lumley: Regression on large data sets: big(g)lm and other approaches
- Erich Neuwirth: R in spreadsheets, RExcel and a little bit of ROO
- Jim Porzak: Customer Subscription Survival in R
- Antony Unwin: Graphical Exploratory Data Analysis
- Jing Hua Zhao: Genome-wide Association Studies
- Thomas Baier: Embedding R in Custom and Standard Applications
- Karim Chine: A federative, collaborative and R-based platform for e-Science, an overview
- Andrea S. Foulkes: Applied Statistical Genetics with R
- Frank E Harrell Jr: Statistical Presentation Graphics
- Thomas Lumley: Survey analysis in R
- Heather Turner and David Firth: Introduction to Generalized Nonlinear Models in R
- Hadley Wickham: Analysing large data with many models
- Simon Wood: GAMs and other smooth GLMs with R

After the official presentations, Rennes, the capital of Brittany offers nice places, pubs and restaurants where you could have interesting discussions and enjoy yourself. Rennes is also famous for its architecture (the Parlement de Bretagne, private mansions, ...), its night life (Festival des tombées de la nuit) and its cuisine (the Festival Gourmand, the Lices Market, ...).

Registration

A web page offering more information on 'useR! 2009', registration, accomodation is available at http://www.R-project.org/useR-2009/

We hope to meet you in Rennes!

The organizing committee:
David Causeur, Julie Josse, Francois Husson, Maela Kloareg, Sébastien Lê, Eric Matzner-Løber and Jérôme Pagès
useR-2009@R-project.org

Conference Review: The 1st Chinese R Conference

and R speaks Chinese

by Yihui Xie

The first Chinese R conference took place at the Renmin University of China (RUC), Beijing, China, December 13 – 14, 2008. The conference was organized and sponsored by the Center for Applied Statistics, RUC, and co-organized by the School of Statistics, RUC. Yihui Xie served as the chair of the conference and program committee; Qingping Zhang and Haoyu Yu were in charge of local arrangements.

Due to the lack of communication among Chinese R users in the past as well as increasing need of using and learning R for users in China, this pioneer meeting mainly focused on

- introducing and popularizing R as a powerful tool for statistical computation and graphics;
- gathering Chinese R users and promoting communication between different disciplines and industries;

Nearly 150 participants from 70 institutions all over China met in Beijing and heard 25 talks in the two-day conference, among which we were honored to meet several Chinese pioneer useRs such as Prof Xizhi Wu (who was the first person introducig R in the School of Statistics, RUC more than 7 years ago) and Guohui Ding (who contributed a lot of work in translating R manuals). Besides, we have also gained various support from overseas leading R/S people such as Richard Becker, John Maindonald (who was invited and tried to give a remote talk at the conference via Skype!) and Kurt Hornik.

The conference program included 13 sessions on several topics:

Opening introduction of the first R conference by Yihui Xie and opening address by Prof Xizhi Wu;

Introduction to R history and development of R by Guohui Ding and Yihui Xie respectively, and R basics by Sizhe Liu;

Statistics Basics survey data analysis by Peng Zhan and statistical simulation by Tan Xi;

Biostatistics introduction to Bioconductor and its application in bioinformatics by Gang Chen, application of R in genetics by Liping Hou, and research on biological competition by Hong Yu;

Data Mining data mining with R by John Maindonald and Yihui Xie, introduction to various data mining methods in R by Sizhe Liu, and R in business intelligence by Jian Li;

Statistical Models quantile regression in R by Chen Zuo;

Bayesian Statistics introduction to Bayesian statistics in R by Peng Ding;

Statistical Computation optimization in R by Taiyun Wei, and simulation and inference of stochastic differential equations using R by Yanping Chen;

Software Development R web applications by Chongliang Li, and integration of R with Microsoft Office via R (D)COM server by Jian Li;

R in Industries visualization of complex system by Xiang Zhang, fitting and projections of mortality stochastic models based on R and Excel by Yunmei Weng, and statistics and R in semiconductor industry by Ming Liu;

Teaching introduction to the R package **animation** by Yihui Xie;

Kaleidoscope application of R on hydrological modeling by Huaru Wang, turning from SAS to R by Jian Wang, using R in Quantitative Structure-Activity Relationship by Bin Ma, exploring irregular data with R by Yihui Xie;

Publication discussion of publishing R-related materials in China by Xiaojie Han;

Participants were really amazed to see the wide applications of R and made warm discussions after each presentation; session chairs often tried hard to control the time for discussion. This has reflected the strong need of learning and using R for Chinese users.

In the end, we decided to make future efforts on

- filling the gaps between statistics and other disciplines with the help of R;
- localization of R, e.g. form a special group to translate more materials on R;
- publishing more Chinese books and papers on R so that users could get started more easily (after this conference, there will be proceedings for publication);

All slides and pictures can be found in this Chinese web page: http://cos.name/2008/12/1st-chinese-r-conference-summary/. This conference was partially supported by the website http://cos.name, which is one of the largest statistical websites and main places for R help in China (see

the board "S-Plus & R" in the forum). We look forward to the next R conference in China and warmly welcome more people (including overseas friends) to attend it.

Yihui Xie
School of Statistics, Room 1037, Mingde Main Building, Renmin University of China, Beijing, 100872, China
xieyihui@gmail.com

Changes in R 2.9.0

by the R Core Team

Changes in R 2.9.0 patched release

New features

- new `anyDuplicated(x)` returns 0 (`= FALSE`) or the index of the first duplicated entry of `x`.
- `matplot()`, `matlines()` and `matpoints()` now also obey a 'lend' argument, determining line end styles. (Wish of PR#13619.)

Bug fixes

- The '`...`' argument was not handled properly when '`...`' was found in the enclosure of the current function, rather than in the function header itself. (This caused `integrate()` to fail in certain cases.)
- `col2rgb("#00000080", TRUE)` would return the background colour. (Reported by Hadley Wickham.)
- `interaction()` now ensures that its result's levels are unique.
- `packageDescription()` and hence `sessionInfo()` now report the correct package version also for a non-attached loaded namespace of a version different from the default `lib.loc`.
- `smoothScatter()` now also works when e.g. `xlim[2] < xlim[1]`.
- `parse_Rd()` would mishandle closing braces when they occurred at the start of a line within an R string in an 'Rd' file. (Reported by Alex Couture-Beil.)
- `readNEWS()` missed version numbers with more than one digit.
- building R '`--without-x`' no longer fails (PR#13665).
- `printCoefmat(cbind(0,1))` now works too (PR#13677).

Changes in R 2.9.0 initial release

Significant user-visible changes

- `expand.grid()` by default no longer coerces strings into factors.

New features

- Package **Matrix** is now a recommended package contained in the basic R distribution. It provides S4 classes and methods for dense and sparse matrices, often by using the numerical libraries Lapack and from the SuiteSparse collection CHOLMOD, CSparse, and SPQR among others.
- `pdf()` and `postscript()` gain a `useKerning` argument to place strings using kerning (which had previously been ignored in display but not in `strwidth`), based in part on an idea and code from Ei-ji Nakama. The default is `TRUE`.

  Kerning involving spaces is now ignored (it was previously only used in the computation of string widths).
- `seq.default()` and `seq.int()` ensure that the result is within the interval `[from, to]` even when the last value would previously have been slightly out of range because of the allowance for rounding error in computing the number of elements.
- `boxplot()` gains a simple `"matrix"` method, enabling `boxplot(mat)` instead of `boxplot(data.frame(mat))`.
- `zip.file.extract()` gains an optional `dir` argument (but use `unzip()` instead).
- `source()` with `encoding` specified (and not as "unknown") marks the encoding of character strings in Latin-1 and UTF-8 locales.
- `parse(text=z)` now marks the encoding of character strings in Latin-1 and UTF-8 locales if `z` is of known encoding (that is all elements are either ASCII or marked as Latin-1 or UTF-8).
- `sprintf()` does stricter error checking on input formats to avoid passing invalid formats to the OS (which have a tendency to crash under such inputs).
- `expand.grid()` gains a `stringsAsFactor` argument to ask for automatic conversion of character vectors to factors (which happened for many years but was not previously documented, and is no longer the default).
- `bxp()` now computes the `ylim` including the outliers only if `outline = TRUE`. (Wish of PR#13196.)
- `barplot()` gains a `args.legend` argument. (Wish of PR#13265.)

- `RweaveLatexSetup()` now accepts all (and not just some) options passed through from `Sweave()`.
- `cumsum(x)` and `cumprod(x)` for double precision `x` now use a long double accumulator where available and so more closely match `sum()` and `prod()` in potentially being more accurate.
- `plot()` methods for `"stepfun"` (and hence `"ecdf"`) gain a `col` argument, allowing to set all three colors simultaneously.
- Iterating over a factor in a for loop now coerces to a character vector (rather than using the integer codes).
- `data.frame()` now recycles columns of list arguments, not just vectors and factors.
- `plot.ts(plot.type="multiple")` now makes use of `*.lab` and `*.axis` graphical parameters (wish of PR#13134 and PR#13135).
- Classes can be exported from a name space using the 'NAMESPACE' file directive '`exportClassPattern`' which has the same syntax as '`exportPattern`'.
- `strftime()` now converts its first argument with `as.POSIXlt()` and so is no longer an alias for `format.POSIXLt`.
- `body<-()` now treats list values consistently with other types: they no longer need to be wrapped in a `list()` call.
- `option("pdfbrowser")` is now set on Windows as well as on Unix-alikes.
- `object.size()` now returns an object of class '`object_size`' and has a `print()` method.
- `[col/row]Sums()`, `*Means()` now have an additional '`...`' argument, so that they can more easily be turned into generic functions.
- Package **tools** contains `dependsOnPkgs()` to compute reverse dependencies.
- Strict type checking is now turned on: this catches more internal corruptions, and some badly written contributed code.
- There are new functions in package **tcltk**, `tk_choose.files()`, `tk_choose.dir()` and `tk_messageBox()`, analogues of functions available on Windows (the last is an analogue of winDialog).
- `Sys.glob()` now does tilde expansion on all platforms.
- `read.table()` and friends gain a `fileEncoding` argument to make re-encoding of input just a little bit simpler.
- `grep()` gains an `invert` argument mimicking '`grep -v/--invert`'.
- `strwrap()` now allows a separate prefix for the first line.
- `grep()` has a more efficient sibling `grepl()` that returns a logical vector.
- `xfig()` has new arguments `defaultFont` and `textSpecial` contributed by Sebastian Fischmeister.
- `parse()` and `parse_Rd()` now point to syntax errors in the reported error context, and include the filename and line and column numbers so smart text editors can jump to the error location.
- `str(<1d-array>)` now writes `"[1:n(1d)]"` instead of the previous less clear `"[, 1:n]"`.
- New function `testInstalledPackage()` in package **tools** allows the examples (and if they were installed) any package-specific tests to be run on an installed package.

  `testInstalledPackages()` can run all the examples and tests in the standard and/or recommended packages. Also `testInstalledBasic()` can run the basic tests (if installed).
- `file.copy()` now has a `recursive` argument.
- Errors in `setOldClass()` will cause a previous definition to be restored.
- Ambiguities in class inheritance and method selection resulting from duplicates in superclasses are now resolved by requiring (if possible) consistency with all the superclass inheritance. The rules for method selection have been revised to take advantage of the improved ordering. See `?Methods` and the reference there related to inheritance.
- New function `unzip()` in package **utils** to expand or list zip archives.
- Replacement functions for `class()` and `oldClass()` will unset the S4 bit when the replacement can't be an S4 object; `oldClass()` will return the S3 class for S4 objects with slot `.S3Class`.
- `clip()` takes extra steps to avoid the next graphics call resetting the clip region.
- New function `sample.int()` to provide documented access to the internal part of `sample()` (sampling from `seq_len(n)`).

- New version of function `withVisible()` for better handling of cases like `withVisible(eval.parent(...))`. Moved to package **base** with a view to replace `.Internal(eval.with.vis)` in `source()` later.

- `showClass()` which is also used to auto-print class definitions, now mentions the package where the class comes from, if there is one.

- `simulate(obj)` now also works for `"glm"` objects and for weighted fits, thanks in part to contributions from Ben Bolker and Heather Turner. There is now a means to extend the methods available for `"glm"` objects, as `glm` families can have an optional `simulate` component.

- S4 classes that inherit from any of the `"structure"` classes or from `"vector"` will turn on methods for all the 'Ops' group of functions when the package containing the classes is loaded. See `class?structure`.

- A mechanism now allows S4 classes to inherit from object types `"environment"`, `"externalptr"` and symbol (`"name"`). See `?setClass`.

- `demo()` gains `echo` and `ask` arguments, with defaults similar to `example()`.

- `library()` no longer checks for the packages merged during the re-organization of 1.9.0.

- New function `poisson.test()` in package **stats** for exact test of rates and rate ratios.

- New function `isdebugged()` indicates whether its argument has the debug flag set or not.

- `ls.str()` [via `print` method] now also works when some objects in the environment are `missing()`.

- Subsetting S4-objects (without an explicit `"["` method) no longer preserves the class in cases like `setClass("C", contains="list");` This reverts a "bug fix" activated in R 2.8.0.

- `.packages()` and `.find.packages()` no longer check the package info for installed packages with dumped metadata, since this was checked when the package was installed. `.packages()` only considers such packages to be validly installed (any others were installed in a long-obsolete version of R). Both changes speed up searches in libraries of thousands of packages.

- `boxplot()` uses butt line endings for its median line (suggestion of Uwe Ligges, PR#13553).

- S4 objects passed to a non-default S3 method will be converted to a valid S3 object with the S3 class. See the section on inheriting from non-S4 classes in `?Classes`.

- A new class `"nonStructure"` has been defined; classes that extend a vector class but that should lose their slots under 'Math' or 'Ops' functions should extend this class. See `class?nonStructure`.

- `axis.POSIXct()` now plots in the timezone marked for its inputs (if any) rather than in the local time. The latter was a deliberate choice, but is easy to select by removing the `tzone` attribute. (Suggestion of Dan Kelley.)

- A new function `classesToAM()` returns an adjacency matrix representing the inheritance of the classes specified. Allows better human examination of the patterns, e.g. by using the matrix as input to one of the graph packages (see the documentation).

- `X11options(antialias = "none")` now works, for consistency with `X11()`.

- `sprintf()` now allows zero-length arguments (with a zero-length result). (Suggestion of Bill Dunlap.)

- `unlink()` is now able to remove broken symbolic links on Unix-alikes.

- New `selectSuperClasses()` utility in package **methods**.

- `HoltWinters()` now allows parameters `alpha` and `beta` to be fixed at 0 and hence `beta = FALSE` and `gamma = FALSE` are used to specify restricted models.

- A new function `smoothScatter()` has been added to package **graphics**. It is appropriate for very dense scatter plots and uses density estimation and color to reflect density of plotting.

Deprecated & defunct

- `allGenerics()` is defunct.

- Use of 'allocVector(CHARSXP ...)' is defunct and gives an error.

- The compatibility define for graphics structure 'NewDevDesc' in 'GraphicsDevice.h' has been removed.

- Support for versioned installs ('R CMD INSTALL --with-package-versions' and `install.packages(installWithVers = TRUE)`) has been removed. Packages installed with versioned names will be ignored.

- The numeric and `power(0.5)` forms of argument to `make.link()` which were deprecated in 2.4.0 are now defunct: use `power()` directly.
- Conversion to 'Sd' and 'Ssgm' by 'R CMD Rdconv' is now defunct.
- Support for 'R --gui=gnome' is now defunct (and package **gnomeGUI** has been withdrawn as it used a long-obsolete version of GNOME).
- 'R CMD SHLIB' on Windows will call the first target (not 'all') in 'Makevars[.win]' in future versions: so make 'all' the first target if you have any.

Utilities

- 'R CMD build' now also uses a 'Makevars[.win]' file for cleaning up 'src/'.
- 'R CMD Rd2dvi' and 'R CMD check' are now able to cope with Cyrillic characters in UTF-8 if environment variable _R_CYRILLIC_TEX_ is set to a non-empty value and the LaTeX system has suitable fonts (thanks to a hint from Alexey Shipunov).
- New function `rtags()` in package **utils** that provides etags-like indexing capabilities for R code files.

  New front-end script 'R CMD rtags' provides an interface to the `rtags()` function (see 'R CMD rtags --help' for details).
- New environment variable `R_TEXI2DVICMD` to hold the path (if any) to texi2dvi found at configure time: this now provides the default to `option("texi2dvi")`.
- 'massage-Examples.pl' has been replaced by the R function `tools:::massageExamples()`.
- 'R CMD REMOVE' now uses `remove.packages()` and hence removes all members of a bundle.
- 'R CMD SHLIB' is now an R script and has a new option '-n' aka '--dry-run' to show what commands would be run. The same code is used on Unix and Windows.
- 'R CMD Rdconv' has new options '--package' and '--version' to set the corresponding fields in HTML conversion.
- 'R CMD check' runs the package tests with a custom startup file, currently containing 'options(useFancyQuotes = FALSE)'.

  Those tests are run by an R script: using a 'tests/Makefile' (undocumented) no longer works.
- 'R CMD config' now knows about 'DYLIB_EXT' and 'SHLIB_EXT', for use in configure files.
- 'R CMD BATCH' has a new option '--no-timing' to suppress printing out the session timing.
- 'R CMD Rd2dvi' can now work on an installed package.
- 'R CMD check' no longer loads package **tcltk** when checking for code problems, so more problems may be reported.
- For 'R CMD SHLIB' on Windows the default 'all' target only makes the DLL, and no longer call targets 'before' and 'after'.

Rd conversion changes

- 'Rd' files have an optional `\Rdversion{}` section, which if missing defaults to 1.0. There is support for version 1.1, a slightly modified version with the following changes:

  - The warnings for `\code{}` inside `\examples` are suppressed.

  - Whitespace between arguments in `\item` and `\section` is accepted without a warning (see below).

  - `$` is treated literally in text, even for latex conversions.

  - `\` is only an escape before `% { } \`.

  - `\R`, `\dots` and `\ldots` can be followed by `{}`, and it is recommended that they are when not followed by whitespace.

  - The obsolete interpretation of `\Alpha` etc is no longer done.
- 'Rd' conversion now handles `^ ~ < > |` correctly in non-code environments (such as `\samp`), and # and _ even in latex conversion (but `$` still needs to be escaped in version 1.0).
- Whitespace between first and second arguments is now accepted for `\item` and `\section`, e.g. `\item{foo}` some value. Previously arguments after whitespace were silently ignored, and a warning is given for version 1.0 files.
- The 'Rd' files created by `prompt()` and friends are declared to be version 1.1.
- `\alias` now supports the escaping of `{` as well as of `%`, and this is recommended.
- `parse_Rd()`, an experimental parser for 'Rd' files, and `Rd2txt()`, `Rd2HTML()`, `Rd2latex()` and `Rd2ex()`, even more experimental converters, have been added to package **tools**.

- 'R CMD check' runs the package's 'Rd' files through parse_Rd() for a stricter syntax check. This can be suppressed by setting _R_CHECK_RD_PARSE_ to FALSE.

- Added markup \verb, which displays like \code, but parses as verbatim text. Currently only supported by parse_Rd() and Rd2HTML().

Installation changes

- The shell used by the 'R' script and other shell scripts intended to be run directly can be specified at installation time by setting the (precious) configure variable R_SHELL.

- libtool has been updated to 2.2.6a.

- '--with-ICU' is now the default: this means that ICU will be used for collation on Mac OS >= 10.4.

- 'make install-tests' can be used to install the test files, to allow an installed version of R to be tested – see the 'R-admin' manual. This is also supported by the function testInstalledPackages() in package **tools**.

- 'make install' using a parallel make should now work.

- 'make check' now always re-makes and re-runs the package examples, which are now collated in the locale's order (and not ASCII order).

- configure will now set the default optimization level for gfortran on x86_64 Linux to '-O' as '-O2' has caused problems with gfortran 4.3.x.

Package installation changes

- install.packages() is able to infer that repos=NULL was intended from the extension on the file name specified as pkgs.

  On Mac OS X it now supports local binary packages with '.tar.gz' extension. Nonetheless '.tgz' remains the preferred extension and is expected in repositories.

  It now checks >= version dependencies for dependent packages, and so will install a newer version of a dependency if needed and available on the repositories.

  The library being installed into is considered when looking for installed packages if it is not already part of .libPaths() (as 'INSTALL' already does).

  It has a new argument Ncpus to support parallel installs of source packages.

- HTML links will be resolved first to the standard packages: this avoids other installed packages diverting help on e.g. qr() and plot() to themselves. The HTML files are only "touched" if they have changed.

- A check is done that the R files can be parsed: this both prevents a broken package without lazy-loading from being installed and gives better diagnostics.

- install.packages() gains a configure.vars argument, and both this and configure.args get their defaults from options().

- There is a unified R script for 'INSTALL' on both Unix-alike and Windows that takes option names used by either in the past.

  It adds '--no-multiarch' to disable building other than the main sub-architecture, and allows multiple instances of '--configure-args' and '--configure-vars' (which will be concatenated).

  New option '--install-tests' will install any package-specific tests.

- Times in the 'Packaged:' and 'Built:' fields are now recorded in UTC, and in most cases in ISO 8601 format.

C-level facilities

- A helper function, asCharacterFactor, converts from factors to character vectors.

Bug fixes

- The postscript() output for setting text is faster and smaller.

- Subsetting a data frame with duplicate column names without selecting columns (e.g. z[i,]) no longer makes the column names unique. This was never documented, but some packages have assumed it.

- data.frame() no longer ignores row names on objects if the first name is empty. (PR#13230: this has been the behaviour for a long time, but apparently undocumented.)

- deparse(control="S_compatible") now never uses backticks.

- X-spline drawing is improved for cases where the control points are located well off the edges of the device.

  The symptom of this problem is the error "reached MAXNUMPTS".

- exists() with 'mode= "any"' will no longer run an active binding's function.

- `format(c(1 + 2i, NA))` no longer has extraneous space in `" NA"`.
- `mood.test()` could fail in 2.8.x on large samples because of integer overflow.
- `heatmap()` without a dendrogram could fail (PR#13512).
- Checks for missing values will no longer occasionally result in an infinite loop or stack overflow error, depending on the compiler. Active bindings are now always considered to be non-missing.
- 'Rd' conversion was not accepting `\p` (as in `\pkg`) or (when using Perl 5.10.x) `\k` (as in `\kbd`) in any preamble text in a section, since those are nowadays interpreted by Perl (PR#13575).
- `if(as.raw(1)) TRUE` now works as expected (PR#13630). Also, `c(as.raw(12), TRUE)` or `c(raw(3), pi)` do.
- `duplicated(<data frame>, incomparables = NA)` now gives the intended error message (PR#13632).
- Name handling of `as.data.frame()` has been sanitized somewhat.
- Evaluating an assignment expression with a string on the left hand side no longer destructively changes the string to a symbol in the expression.

Changes on CRAN

by Kurt Hornik

Publishing a source package on CRAN now adds the publication date and the repository name (CRAN) to the package 'DESCRIPTION', and standardizes the license specification if possible.

CRAN package web

The CRAN package web pages (remember that these gained persistent URLs of the form

`http://CRAN.R-project.org/package=`*foo*

in early 2008) have again been enhanced.

If available, package citation (from 'inst/CITATION', suitably HTMLified), installation (from a top-level 'INSTALL' file), classification ('DESCRIPTION' Classification/ACM, Classification/JEL and Classification/MSC fields with subject classifications using the Computing Classification System of the Association for Computing Machinery, the Journal of Economic Literature Classification System, and the Mathematics Subject Classification of the American Mathematical Society, respectively) and language ('DESCRIPTION' Language field) information is now linked from the package web page.

In addition, the direct reverse dependencies of the package (i.e., the CRAN packages which depend on, import, suggest or enhance the package) are shown at the bottom of the package web page.

Standardizable license specifications are fully hyperlinked to the respective license texts.

All CRAN (HTML) web pages should now be valid XHTML.

New contributed packages

AIGIS Areal Interpolation for GIS data. Can be used to interpolate spatially associated data onto arbitrary target polygons which lack such data. Version 1.0 of the package is oriented toward convenient interpolation of specific US census data for California, but the tools provided should work for any combination of GIS data source and target polygon, provided appropriate care is taken. Future versions will be aimed at facilitating more general applications. By Benjamin P. Bryant and Anthony Westerling.

Animal Functions for analyzing animal (including human) behavior data originating from a variety of sources, including functions to analyze time coded behaviors from CowLog (open source software for coding behaviors from digital video) data files and observation files with similar format. Other features include hourly, daily, weekly and monthly summaries of time coded data, analysis of RIC (roughage intake system, Insentec automation) data files and labeling measurement data according to behavioral observations for e.g. model building purposes. By Matti Pastell.

AquaEnv An integrated development toolbox for aquatic chemical model generation focused on (ocean) acidification and CO_2 air-water exchange. Contains all elements necessary to model the pH, the related CO_2 air-water exchange, as well as aquatic acid-base chemistry in general for an arbitrary marine, estuarine or freshwater system. Chemical batches can be modeled as well. In addition to the routines necessary to calculate desired information, **AquaEnv** also contains a suite of tools to visualize this. Furthermore, **AquaEnv** can not only be used to build dynamic models of aquatic systems, but can also serve as a simple desktop tool for the experimental aquatic chemist to generate and visualize all possible derived information from a set of measurements with one single easy to use R function. By Andreas F. Hofmann, Karline Soetaert, and Filip J. R. Meysman.

BAMD Bayesian association model for genomic data with missing covariates. Fits a linear mixed model where the covariates for the random effects have missing values. By V. Gopal, Z. Li and G. Casella.

BAS Bayesian model averaging using Bayesian Adaptive Sampling. Performs BMA in linear models using stochastic or deterministic sampling without replacement from posterior distributions. Prior distributions on coefficients are from Zellner's g-prior or mixtures of g-priors corresponding to the Zellner-Siow Cauchy Priors or the Liang et al. hyper-g priors. Other model selection criterion include AIC and BIC. Sampling probabilities may be updated based on the sampled models. By Merlise Clyde, with contributions from Michael Littman.

BGSIMD Block Gibbs Sampler with Incomplete Multinomial Distribution. Implements an efficient block Gibbs sampler with incomplete data from a multinomial distribution taking values from the k categories $1,2,\ldots,k$, where data are assumed to miss at random and each missing datum belongs to one and only one of m distinct non-empty proper subsets $A_1, A_2, \ldots, A_m$ of $1,2,\ldots,k$ and the k categories

are labeled such that only consecutive A's may overlap. By Kwang Woo Ahn and Kung-Sik Chan.

BMN Approximate and exact methods for pairwise binary Markov models. The exact method uses an implementation of the junction tree algorithm for binary graphical models. By Holger Hoefling.

BSagri Statistical methods for safety assessment in agricultural field trials. By Frank Schaarschmidt.

BayesDA Functions for Bayesian Data Analysis, with data sets from the book "Bayesian Data Analysis" (second edition) by Gelman, Carlin, Stern and Rubin. (Not all data sets yet.) Compiled by Kjetil Halvorsen.

BayesX R Utilities Accompanying the BayesX software package for structured additive regression. Provides functionality for exploring and visualizing estimation results obtained with BayesX. Also provides functions that allow to read, write and manipulate map objects that are required in spatial analyses performed with BayesX. By Thomas Kneib, Felix Heinzl, Andreas Brezger, and Daniel Sabanes Bove.

CADFtest Performs Hansen's (1995) Covariate-Augmented Dickey-Fuller (CADF) test. The p-values of the test are computed using a procedure proposed in Costantini, Lupi and Popp (2007), illustrated in Lupi (2008). By Claudio Lupi.

CHNOSZ Calculation of the standard molal thermodynamic properties and chemical affinities of reactions between and among minerals, inorganic, organic, and/or biomolecular aqueous and crystalline species. Incorporation of an extensive database of thermodynamic properties of species and algorithms for computing the standard thermodynamic properties of neutral and ionized proteins from amino acid group contributions. Generation of chemical speciation and predominance diagrams for stable and metastable equilibrium in open systems as a function of temperature, pressure, and chemical activities or fugacities of basis species. By Jeffrey M. Dick.

ClinicalRobustPriors Robust Bayesian priors in clinical trials. Fuquene, Cook, & Pericchi (2008) (http://www.bepress.com/mdandersonbiostat/paper44) make a comprehensive proposal putting forward robust, heavy-tailed priors over conjugate, light-tailed priors in Bayesian analysis. The behavior of Robust Bayesian methods is qualitatively different from Conjugate and short tailed Bayesian methods and arguably much more reasonable and acceptable to the practitioner and regulatory agencies. This package is useful to compute the distributions (prior, likelihood and posterior) and moments of the robust models: Cauchy/Binomial, Cauchy/Normal and Berger/Normal. Both Binomial and Normal Likelihoods can be handled. Furthermore, allows assessment of the hyper-parameters and posterior analysis. By A. Jairo and P. Fuquene.

ConvCalendar Converts between the `"Date"` class and d/m/y for several calendars, including Persian, Islamic, and Hebrew. By 'Project Pluto' (www.projectpluto.com/calendar) and Thomas Lumley.

CvM2SL1Test L_1 version of Cramer-von Mises two sample tests. Contains functions for computing the Cramer-von Mises two sample test scores and the exact p-value(s) for given test score(s) under the assumption that the populations under comparison have the same probability distribution. The L_1 Cramer-von Mises test, like its L_2 counterpart, is distribution-free, but of less computational intensity. In certain cases, this version of Cramer-von Mises test is almost as powerful as its L_2 counterpart. Simulation studies also show that it is more powerful than the Kolmogorov-Smirnov test in certain cases. By Yuanhui Xiao and Ye Cui.

DAKS Data Analysis and Knowledge Spaces. Functions and example data sets for the psychometric theory of knowledge spaces. Implements data analysis methods and procedures for simulating data and transforming different formulations in knowledge space theory. By Anatol Sargin and Ali Ünlü.

DSpat Spatial modeling for distance sampling data. Provides functions for fitting spatial models to line transect sampling data and to estimate abundance within a region. By Devin Johnson, Jeff Laake, and Jay VerHoef.

DTK Functions for conducting and plotting Dunnett's modified Tukey-Kramer pairwise multiple comparison test accounting for unequal variance and unequal sample sizes. By Matthew K. Lau.

Depela Implements semiparametric estimation of copula models, and deals with structural break problems in copula modeling. By Andrew C. Chou and Jing Tao.

EMJumpDiffusion Estimate parameters for Jump Diffusion processes via the EM algorithm. The jump times are considered to be Bernoulli distributed with normally distributed jump sizes. By Matthias Graser.

EngrExpt Data sets from Nelson, Coffin and Copeland "Introductory Statistics for Engineering Experimentation" (Elsevier, 2003) with sample code. R port by Douglas Bates and Karen A. F. Copeland.

ExPD2D Exact computation of bivariate projection depth based on Fortran code. By Yijun Zuo and Xiangyang Ye.

FBN FISH Based Normalization and copy number (CN) inference of SNP microarray data. Normalizes the data from a file containing the raw values of the SNP probes of microarray data by using the FISH probes and their corresponding CNs. By Adrian Andronache and Luca Agnelli.

FD Measuring functional diversity (FD) from multiple traits. Computes different multidimensional FD indices. Implements a distance-based framework to measure FD that allows any number and type of functional traits, and can also consider species relative abundances. By Etienne Laliberti£¡.

FKF Fast Kalman Filter. A flexible implementation entirely written in C and fully relying on linear algebra subroutines contained in BLAS and LAPACK. Due to the speed of the filter, the fitting of high-dimensional linear state space models to large data sets becomes possible. Also contains a function for the visualization of the state vector and graphical diagnostics of the residuals. By David Lüthi and Philipp Erb.

FSelector Functions for selecting attributes from a given data set. Attribute subset selection is the process of identifying and removing as much of the irrelevant and redundant information as possible. By Piotr Romanski.

FactoClass Multivariate exploration of a data table with factorial analysis and cluster methods. By Campo Elías Pardo and Pedro César del Campo, with contributions from Ivan Diaz and Mauricio Sadinle.

Formula Infrastructure for extended formulas (e.g., with two parts on the right hand side). By Yves Croissant and Achim Zeileis.

GFMaps Visualization technique for interpretation of high-throughput genetic or proteomic experiments. Integrates data sets derived from gene expression profiles with preexisting information from public databases such as KEGG pathway or Gene Ontology using Gene Set Enrichment Analysis (GSEA) and introduces a framework allowing to visualize biologically annotated data sets and relevant ratings of genes via a color gradient. By Sanjay Bhatikar and Kanika Arora.

GRRGI Gauge R and R Confidence Intervals. Generates confidence intervals for the variance components in Gauge R and R data using ANOVA with the Satterthwaite approximation as well as the method of Generalized Inference. By Walter Resch, with information from "Measurement System Assessment Via Generalized Inference" by Michael Hamada and Sam Weerandi.

GridR Executes functions on remote hosts, clusters or grids. In addition, users are provided with an interface to share variables and functions with other users. By Dennis Wegener, Malte Lohmeyer and Stefan Rüping.

HadoopStreaming A framework for writing map/reduce scripts for use in Hadoop Streaming. Also facilitates operating on data in a streaming fashion, without Hadoop. By David S. Rosenberg.

HaploSim Simulate haplotypes through meioses. Allows specification of population parameters. By Albart Coster, John Bastiaansen.

LIM Functions that read and solve Linear Inverse Model (food web, linear programming) problems, which find solutions to linear or quadratic functions: min or max $f(x)$, where $f(x) = ||Ax - b||^2$ or $f(x) = \sum a_i x_i$ subject to equality constraints $Ex = f$ and inequality constraints $Gx \geq h$. Uses package **limSolve**. By Karline Soetaert and Dick van Oevelen.

LambertW Lambert W parameter estimation, plots, simulation. Lambert W random variables offer a new way of dealing with slightly skewed data. By Georg M. Goerg.

LearnEDA Functions for Learning Exploratory Data Analysis. By Jim Albert.

MAMSE Calculation of the nonparametric Minimum Averaged Mean Squared Error (MAMSE) weights for univariate, right-censored or multivariate data. The MAMSE weights can be used in a weighted likelihood or to define a mixture of empirical distribution functions. By Jean-François Plante.

MCE Tools for evaluating Monte Carlo Error. Has functions to estimate MC error in simulation studies, and functions helping with planning the number of replications required in a simulation study to reach a specified level of certainty. By Elizabeth Koehler and Sebastien Haneuse.

MCMCglmm MCMC Generalized Linear Mixed Models. By Jarrod Hadfield.

MixSim Simulate data to study performance of clustering algorithms. By Volodymyr Melnykov, Wei-Chen Chen and Ranjan Maitra.

ModelMap Create random forest and stochastic gradient boosting models, and apply them to GIS '.img' files to build detailed prediction maps. Validates the models with an independent test set, cross validation, or in the case of random forest Models, with out of bag (OOB) predictions on the training data. Creates graphs and tables of the model validation results, and applies the models to GIS '.img' files of predictors to create detailed prediction surfaces. Handles large predictor files for map making via chunking. By Elizabeth Freeman and Tracey Frescino.

Multiclasstesting Performance of N-ary classification testing. By C. Nardini and Y.-H. Liu.

NMRS NMR Spectroscopy. Developed to directly load spectra in the Bruker spectroscopy format. Displays the spectrum reference and manages basic operations such as setting the chemical shift of a certain compound (TSP or DSS) to 0 ppm, zero order and first order phase corrections, baseline adjustment and spectral area selection. By José L. Izquierdo.

OAIHarvester Harvest metadata using the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) version 2.0. By Kurt Hornik.

Oncotree Oncogenetic trees: functions to construct and evaluate directed tree structures that model the process of occurrence of genetic alterations during carcinogenesis. By Aniko Szabo and Lisa Pappas.

OrdFacReg Least squares, logistic, and Cox-regression with ordered predictors. In biomedical studies, researchers are often interested in assessing the association between one or more ordinal explanatory variables and an outcome variable, at the same time adjusting for covariates of any type. The outcome variable may be continuous, binary, or represent censored survival times. In the absence of a precise knowledge of the response function, using monotonicity constraints on the ordinal variables improves efficiency in estimating parameters, especially when sample sizes are small. This package implements an active set algorithm that efficiently computes such estimators. By Kaspar Rufibach.

OrdMonReg Compute least squares estimates of one bounded or two ordered antitonic regression curves. Consider the problem of estimating two antitonic regression curves f_1 and f_2 under the constraint that $f_1 \geq f_2$. Given two sets of n data points $g_1(x_1),\ldots,g_1(x_n)$ and $g_2(x_1),\ldots,g_2(x_n)$ that are observed at (the same) deterministic points $x_1,\ldots,x_n$, the estimates are obtained by minimizing the Least Squares criterion $L(f_1,f_2) = \sum_{i=1}^{n}(g_1(x_i) - f_1(x_i))^2 w_1(x_i) + \sum_{i=1}^{n}(g_2(x_i) - f_2(x_i))^2 w_2(x_i)$ over the class of pairs of functions (f_1,f_2) such that f_1 and f_2 are antitonic and $f_1(x_i) \geq f_2(x_i)$ for all $i = 1,\ldots,n$. The estimates are computed with a projected subgradient algorithm where the projection is calculated using a PAVA. By Fadoua Balabdaoui, Kaspar Rufibach, and Filippo Santambrogio.

PMA Performs Penalized Multivariate Analysis: a penalized matrix decomposition, sparse principal components analysis, and sparse canonical correlation analysis, described in "A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis" by Witten, Tibshirani and Hastie (2009). By Daniela Witten and Rob Tibshirani.

PairViz Visualization using Eulerian tours and Hamiltonian decompositions. By C. B. Hurley and R. W. Oldford.

PhViD Pharmacovigilance signal detection methods extended to the multiple comparison setting. Functions can be used as standard R functions or through an user friendly interface (PhViD.gui). By Ismail Ahmed and Antoine Poncet.

RDS Functionality for carrying out estimation with data collected using Respondent-Driven Sampling. Current functionality is extremely limited. By W. Whipple Neely.

REQS R/EQS Interface. Contains the function `run.eqs()` which calls an EQS script file, executes the EQS estimation, and, finally, imports the results as R objects. These two steps can be performed separately: `call.eqs()` calls and executes EQS, whereas `read.eqs()` imports existing EQS outputs as objects into R. By Patrick Mair and Eric Wu.

RHRV Heart rate variability analysis of ECG data. By M. Lado, A. Méndez, D. Olivieri, L. Rodríguez-Liñares, and X. Vila.

RMTstat Distributions and statistics from Random Matrix Theory. Functions for working with the Tracy-Widom laws and other distributions related to the eigenvalues of large Wishart matrices. The tables for computing the Tracy-Widom densities and distribution functions were computed by Momar Dieng's MATLAB package 'RMLab' (`http://math.arizona.edu/`

~momar/research.htm). By Patrick O. Perry, with contributions from Iain M. Johnstone, Zongming Ma, and Morteza Shahram.

ROptEstOld Optimally robust estimation using S4 classes and methods. Old version still needed for current versions of **ROptRegTS** and **RobRex**. By Matthias Kohl.

RQDA R-based Qualitative Data Analysis, currently only supporting plain text. By Huang Ronggui.

RSiteSearch Alternative interfaces to RSiteSearch. By Sundar Dorai-Raj, Romain François and Spencer Graves.

RSurvey Analysis of Spatially Distributed Data. Processes such data and is capable of error corrections and data visualization. Also provides a graphical user interface. By Jason C. Fisher.

RcmdrPlugin.SurvivalT An **Rcmdr** plug-in based on the **survival** package for easier student access to survival analysis. By Daniel Leucuta.

RcmdrPlugin.orloca An **Rcmdr** plugin providing a GUI for the **orloca** package. By Fernando Fernández-Palacín and Manuel Muñoz-Márquez.

RcmdrPlugin.qcc An **Rcmdr** plug-in based on the **qcc** package, providing an integration between the user and the tools of SPC. By Renan Cortes, with contributions of I. Prestes and Suzi Camey.

RcmdrPlugin.survival An **Rcmdr** plug-in for the **survival** package, with dialogs for Cox models, parametric survival regression models, estimation of survival curves, and testing for differences in survival curves, along with data-management facilities and a variety of tests, diagnostics and graphs. By John Fox.

Rcpp Rcpp R/C++ interface classes and examples. Maps data types between R and C++, and includes support for R types real, integer, character, vector, matrix, `"Date"`, Date-Time (i.e., `"POSIXct"`) at microsecond resolution, data frame, and function. Also supports calling R functions from C++. By Dirk Eddelbuettel, with contributions by Simon Urbanek and David Reiss.

Rcsdp R interface to the CSDP semidefinite programming library. Installs version 6.0.1 of CSDP from the COIN-OR website if required. An existing installation of CSDP may be used by passing the proper configure arguments to the installation command. By Hector Corrada Bravo (CSDP by Brian Borchers).

RgoogleMaps Overlays on Google map tiles in R. Serves two purposes: (i) provide a comfortable R interface to query the Google server for static maps, and (ii) use the map as a background image to overlay plots within R. By Markus Loecher.

Rlabkey Import data from a labkey database into an R data frame. By Valerie Obenchain.

RxCEcolInf $R \times C$ Ecological Inference With optional incorporation of survey information. Fits the $R \times C$ inference model described in Greiner and Quinn (2009). By D. James Greiner, Paul Baines, and Kevin M. Quinn.

SDaA Functions and data sets from "Sampling: Design and Analysis" S. Lohr (1999), Duxbury. By Tobias Verbeke.

SEMModComp Model Comparisons for structural equation modeling (SEM). Conduct tests of difference in fit for mean and covariance structure models as in SEM. By Roy Levy.

SGCS Spatial Graph based Clustering Summaries for spatial point patterns. By Tuomas Rajala.

SMIR Companion to "Statistical Modelling in R" by Aitkin et al (2009), Oxford University Press. By Murray Aitkin, Brian Francis, John Hinde, and Ross Darnell.

ScottKnott Multiple comparison test of means using the clustering method of Scott & Knott. By Enio Jelihovschi, José Cláudio Faria, and Sérgio Oliveira.

SimComp Simultaneous Comparisons (tests and confidence intervals) for general linear hypotheses when there is more than one primary response variable (endpoint). The procedure of Hasler (2009) is applied for differences or ratios of means of normally distributed data. The covariance matrices (containing the covariances between the endpoints) may be assumed to be equal or possibly unequal for the different groups. By Mario Hasler.

SpectralGEM Discovering Genetic Ancestry Using Spectral Graph Theory. By Ann Lee, Diana Luca, Bert Klei, Bernie Devlin, and Kathryn Roeder.

StatFingerprints Processing and statistical analysis of molecular fingerprint profiles. By Rory Michelland and Laurent Cauquil.

Stem Estimation of the parameters of a spatio-temporal model using the EM algorithm, estimation of the parameter standard errors using a spatio-temporal parametric bootstrap, and spatial mapping. By Michela Cameletti.

SubpathwayMiner Annotation and identification of the KEGG pathways. Facilitates sub-pathway annotation and identification of metabolic pathways. Also provides annotation and identification of entire pathways. By Chunquan Li.

SweaveListingUtils Utilities for Sweave together with the TEX listings package. Features defining R/Rd as a listings "language" and including R/Rd source files (snippets) copied from R-Forge in its most recent version (or another URL) thereby avoiding inconsistencies between vignette and documented source code. By Peter Ruckdeschel.

TRIANG Discrete triangular distributions, which complete the classical discrete distributions like binomial, Poisson and Negative binomial. By Tristan Senga Kiessé, Silvio S. Zocchi, and Célestin C. Kokonendji.

TSPostgreSQL Time Series database interface (TSdbi) extensions for PostgreSQL. By Paul Gilbert.

TSfame Time Series database interface (TSdbi) extensions for fame. By Paul Gilbert.

TShistQuote Time Series database interface (TSdbi) extensions for `get.hist.quote` to retrieve data from historical quote URLs. By Paul Gilbert.

TSodbc Time Series database interface (TSdbi) extensions for ODBC. By Paul Gilbert.

TeachingSampling Foundations of inference in survey sampling. By Hugo Andrés Gutiérrez Rojas.

VarianceGamma The Variance Gamma Distribution. Provides density, distribution and quantile functions. In addition, there are also functions for random number generation and fitting of the variance gamma to data. By David Scott and Christine Yang Dong.

WaveCGH Wavelet Changepoint Detection for Array CGH. Detects changepoints and finds gain/loss regions. By M. S. Islam and A. I. McLeod.

WriteXLS Cross-platform Perl based R function to create Excel 2003 ('.xls') files from one or more data frames. Each data frame will be written to a separate named worksheet in the Excel spreadsheet. The worksheet name will be the name of the data frame it contains. By Marc Schwartz.

YourCast Makes time series cross-sectional forecasts with multiple cross-sections based on your assumptions given a variety of smoothing assumptions based on similarities among the levels or trends in the expected value of the dependent variable rather than the coefficients. By Federico Girosi, Gary King, and Elena Villalon.

afc Calculate the Generalized Discrimination Score, also known as Two Alternatives Forced Choice Score (2AFC). By Andreas Weigel.

agreement Investigate agreement between two measurement methods using a simulation approach. By Fabio Frascati and Bruno Mario Cesana.

alphahull Computes the alpha-shape and alpha-convex hull of a given sample of points in the plane. These concepts generalize the definition of the convex hull of a finite set of points. The programming is based on the duality between the Voronoi diagram and Delaunay triangulation. Also includes a function that returns the Delaunay mesh of a given sample of points and its dual Voronoi diagram in one single object. By Beatriz Pateiro-López and Alberto Rodríguez-Casal.

amei Adaptive Management of Epidemiological Interventions. Provides a flexible statistical framework for generating optimal epidemiological interventions that are designed to minimize the total expected cost of an emerging epidemic while simultaneously propagating uncertainty regarding underlying disease parameters through to the decision process via Bayesian posterior inference. The strategies produced through this framework are adaptive: vaccination schedules are iteratively adjusted to reflect the anticipated trajectory of the epidemic given the current population state and updated parameter estimates. By Daniel Merl, Leah R. Johnson, Robert B. Gramacy, and Marc Mangel.

archetypes A framework for archetypal analysis supporting arbitrary problem solving mechanisms for the different conceptual parts of the algorithm. (Used as real-world test application for the Roxygen documentation system.) By Manuel J. A. Eugster.

arulesNBMiner An implementation of the model-based mining algorithm for NB-frequent itemsets presented in "A model-based frequency constraint for mining associations from transaction data" by M. Hahsler (*Data Mining and Knowledge*, 2006). In addition, implements an extension for NB-precise rules. By Michael Hahsler.

ascii Export R objects to asciidoc or txt2tags. Comes with two drivers for Sweave. By David Hajage.

aspect Aspects of Multivariables. Consists of two main functions: `corAspect()` performs optimal scaling by maximizing an aspect (i.e., target function such as sum of eigenvalues, sum of squared correlations, squared multiple correlations, etc.) of the corresponding correlation matrix. `lineals()` performs optimal scaling by minimizing a non-correlational aspect based on pairwise correlations and correlation ratios. The resulting correlation matrix and category scores can be used for further multivariate methods such as SEM. A platform including related **PsychoR** packages is provided on R-forge. By Jan de Leeuw and Patrick Mair.

asympTest Asymptotic testing. By the Cqls Team.

automap Performs automatic interpolation by automatically estimating the variogram and then calling **gstat**. By Paul Hiemstra.

aylmer A generalization of Fisher's exact test that allows for structural zeros. By Robin K. S. Hankin (R) and Luke G. West (C++).

bayesCGH Bayesian analysis of array CGH data. A set of methods for Bayesian analysis of proportions of chromosomal changes within clinical sets. By Thomas Hardcastle.

bayesclust Tests/Searches for significant clusters in genetic data. By V. Gopal, C. Fuentes and G. Casella.

beanplot Visualization via beanplots (univariate comparison graphs providing an alternative to boxplot, stripcharts and violin plots). By Peter Kampstra.

bethel The sample size according to the Bethel's procedure. By Michele De Meo.

binarySimCLF Simulate correlated binary data using an algorithm based on Qaqish (2003). By Kunthel By and Bahjat F. Qaqish.

blockmodeling Generalized and classical blockmodeling of valued networks. In addition, measures of similarity or dissimilarity based on structural equivalence and regular equivalence (REGE algorithm) can be computed and partitioned matrices can be plotted. By Ales Ziberna.

bootspecdens Bootstrap for testing the hypothesis that the spectral densities of a number $m \geq 2$ of not necessarily independent time series are equal. By Tatjana Kinsvater.

canvas R graphics device targeting the HTML canvas element. Implements the CanvasRenderingContext2D JavaScript API. By Jeffrey Horner.

ccems Combinatorially Complex Equilibrium Model Selection. Dissociation constants of quasi-equilibriums of enzymes and protein-ligand binding equilibriums in general are systematically scanned though possibilities of being infinity and/or equal to others. The automatically generated space of models is then fitted to data. By Tom Radivoyevitch.

cellVolumeDist Implements a methodology for using cell volume distributions to estimate cell growth rates and division times as described in "Cell Volume Distributions Reveal Cell Growth Rates and Division Times" by Michael Halter, John T. Elliott, Joseph B. Hubbard, Alessandro Tona and Anne L. Plant (*Journal of Theoretical Biology*). By Katharine M. Mullen, Michael Halter, John Lu and Nathan Dodder.

clinsig Functions for calculating clinical significance. By Jim Lemon.

clues A clustering method based on local shrinking. Contains functions for automatically estimating the number of clusters and obtaining the final cluster partition without any input parameter except the stopping rule for convergence. Also provides functions to evaluate and compare the performances of partitions of a data set both numerically and graphically. By Fang Chang, Vincent Carey, Weiliang Qiu, Ruben H. Zamar, Ross Lazarus, and Xiaogang Wang.

corcounts Generate high-dimensional correlated count random variables with a prespecified Pearson correlation. By Vinzenz Erhardt.

crawl The (C)orrelated (RA)ndom (W)alk (L)ibrary of R functions for fitting continuous-time correlated random walk (CTCRW) models with time indexed covariates. The model is fit using the Kalman-Filter on a state space version of the continuous-time stochastic movement process. By Devin S. Johnson.

depth Depth functions methodology applied to multivariate analysis. Allows calculation of depth values and depth-based location estimators, and includes functions for drawing contour plots and perspective plots of depth functions. By Jean-Claude Massé and Jean-François Plante.

diffusionMap Implements the diffusion map method of data parametrization, including creation and visualization of diffusion map and course-graining using diffusion K-means. By Joseph Richards and Ann Lee.

diseasemapping Calculate SMR's from population and case data. Works with regular data set files and shape files. By Patrick Brown.

dlmap Detection Localization Mapping for QTL. By Emma Huang and Andrew George.

dyad Analysis of dyadic observational data. Contains original implementation of the Gottman-Murray marriage model and revisions using threshold autoregressive models. These programs allow modeling of dyadic observational data, such as interaction between husband and wife. Intended for researchers interested in modeling social behavior. By Tara Madhyastha and Ellen Hamaker.

dynCorr Computes dynamical correlation estimates and percentile bootstrap confidence intervals for pairs of longitudinal responses, including consideration of lags and derivatives. By Joel Dubin, Dandi Qiao, and Hans-Georg Müller.

elec A bizarre collection of functions written to do various sorts of statistical election audits. There are also functions to generate simulated voting data, and simulated "truth" so as to do simulations to check characteristics of these methods. By Luke Miratrix.

emplik2 Empirical likelihood test (two-sample, censored data). Calculates the p-value for a mean-type hypothesis (or multiple mean-type hypotheses) based on two samples. By William H. Barton, under the supervision of Mai Zhou.

exams Sweave-based automatic generation of standardized exams for large-lecture courses, with multiple-choice questions and arithmetic problems. By Achim Zeileis and Bettina Grün.

ez Facilitates the analysis of factorial experiments, including purely within-Ss designs (a.k.a. "repeated measures"), purely between-Ss designs, and mixed within-and-between-Ss designs. The functions in this package provide easy access to descriptive statistics, ANOVA, permutation tests, and visualization of results. By Michael A. Lawrence.

fast Implementation of the Fourier Amplitude Sensitivity Test (FAST), a method to determine global sensitivities of a model on parameter changes with relatively few model runs. By Dominik Reusser.

fechner Functions and example data sets for Fechnerian scaling of discrete object sets. Computes Fechnerian distances among objects representing subjective dissimilarities, and other related information. By Thomas Kiefer and Ali Ünlü, based on original MATLAB source by Ehtibar N. Dzhafarov.

fishmethods Fishery methods and models from Quinn and Deriso (1999), Haddon (2001), and literature. By Gary A. Nelson.

fit4NM Exploratory analysis for pharmacometrics via the NONMEM platform. By Eun-Kyung Lee, Gyujeong Noh, and Hyeong-Seok Lim.

fitdistrplus Help to fit of a parametric distribution to non-censored or censored data. Censored data may contain left censored, right censored and interval censored values, with several lower and upper bounds. By Marie Laure Delignette-Muller, Régis Pouillot, and Jean-Baptiste Denis.

flashClust Implementation of optimal hierarchical clustering. Code by Fionn Murtagh, packaging by Peter Langfelder.

foba Greedy variable selection via forward, backward, and foba sparse learning algorithms for ridge regression, described in "Adaptive Forward-Backward Greedy Algorithm for Learning Sparse Representations". By Tong Zhang.

fpow Compute the noncentrality parameter of the noncentral F distribution for given probabilities of type I and type II error and degrees of freedom of the numerator and the denominator. May be useful for computing minimal detectable differences for general ANOVA models. The program is documented in "On the computation of the noncentral F and noncentral beta distribution" by A. Baharev and S. Kemeny (*Statistics and Computing*, 2008, http://dx.doi.org/10.1007/s11222-008-9061-3). By Ali Baharev.

futile A collection of utility functions to expedite software development. By Brian Lee Yung Rowe.

gcExplorer Graphical Cluster Explorer. Visualize cluster results and investigate additional properties of clusters using interactive neighborhood graphs. By clicking on the node representing the cluster, information about the cluster is provided using additional graphics or summary statistics. For microarray data, tables with links to genetic databases like gene ontology can be created for each cluster. By Theresa Scharl, Ingo Voglhuber, and Friedrich Leisch.

gof Model diagnostics based on cumulative residuals. By Klaus K. Holst.

gogarch Generalized Orthogonal GARCH (GO-GARCH) models. By Bernhard Pfaff.

gputools Provides R interfaces to a handful of common data-mining algorithms implemented in parallel using a mixture of nVidia's CUDA language and cublas library. On a computer equipped with an nVidia GPU these functions

may be substantially more efficient than native R routines. By Josh Buckner, with contributions from Justin Wilson.

graphicsQC Quality Control for Graphics in R. Provides functions to generate graphics files, compare them with "model" files, and report the results. By Stephen Gardiner and Paul Murrell.

grpreg Efficient algorithms for fitting the regularization path for linear or logistic regression models penalized by the group lasso, group bridge, or group MCP methods. The algorithm is based on the idea of a locally approximated coordinate descent. A technical report describing the methods and algorithms is available at `http://www.stat.uiowa.edu/techrep/tr393.pdf`. By Patrick Breheny.

hacks Convenient R Functions. By Nathan Stephens and Vicky Yang.

hash Implements a data structure using R environments similar to hashes in Perl and dictionaries in Python but with a purposefully R flavor. By Christopher Brown.

hexbin Binning and plotting functions for hexagonal bins. Now uses and relies on **grid** graphics and formal (S4) classes and methods. By Dan Carr, ported by Nicholas Lewin-Koh and Martin Mächler.

hyperdirichlet A suite of routines for the hyperdirichlet distribution. By Robin K. S. Hankin.

imprProbEst Minimum distance estimation in an imprecise probability model. The imprecise probability model consists of upper coherent previsions whose credal sets are given by a finite number of constraints on the expectations. The parameter set is finite. The estimator chooses that parameter such that the empirical measure lies next to the corresponding credal set with respect to the total variation norm. By Robert Hable.

influence.ME A collection of tools for calculating measures of influential data for mixed effects models. The basic rationale behind identifying influential data is that when iteratively single units are omitted from the data, models based on these data should not produce substantially different estimates. To standardize the assessment of how influential a (single group of) observation(s) is, several measures of influence are common practice. First, DFBETAS is a standardized measure of the absolute difference between the estimate with a particular case included and the estimate without that particular case. Second, Cook's distance provides an overall measurement of the change in all parameter estimates, or a selection thereof. By Rense Nieuwenhuis, Ben Pelzer, and Manfred te Grotenhuis.

introgress Analysis of introgression of genotypes between divergent, hybridizing lineages, including estimation of genomic clines from multi-locus genotype data and testing for deviations from neutral expectations. Functions are also provided for maximum likelihood estimation of molecular hybrid index and graphical analysis. By Zachariah Gompert and C. Alex Buerkle.

ipptoolbox Uncertainty quantification and propagation in the framework of Dempster-Shafer Theory and imprecise probabilities. By Philipp Limbourg.

irtProb Utilities and probability distributions related to multidimensional person Item Response Models (IRT). By Gilles Raiche.

isotone Active set and generalized PAVA for isotone optimization. Contains two main functions: `gpava()` solves general isotone regression problem using the pool-adjacent-violators algorithm (PAVA). `activeSet()` provides a framework of active set methods for isotone optimization problems. Various loss functions are pre-specified. By Jan de Leeuw, Kurt Hornik, and Patrick Mair.

jointDiag Algorithms to perform approximate joint diagonalization of a finite set of square matrices. Depending on the algorithm, an orthogonal or non-orthogonal diagonalizer is found. These algorithms are particularly useful in the context of blind source separation. By Cédric Gouy-Pailler.

kst Knowledge Space Theory: a set-theoretical framework which proposes mathematical formalisms to operationalize knowledge structures in a particular domain. Provides basic functionalities to generate, handle, and manipulate knowledge structures and knowledge spaces. By Christina Stahl and David Meyer.

labeltodendro Convert labels or tables to a dendrogram, offering a dendrogram representation of series of labels as especially needed in Markov Chain Monte Carlo clustering. By Vahid Partovi Nia, Anthony Davison and Arpit Chaudhary.

latticist A graphical user interface for exploratory visualization. Primarily an interface to the Lattice graphics system, but also produces displays from the **vcd** package for categorical data. Given a multivariate data set (either a

data frame or a table), it attempts to produce useful displays based on the properties of the data. The displays can be customized by editing the calls used to generate them. By Felix Andrews.

lcd Learn Chain graphs (and as a special case, Bayesian networks) via Decomposition. By Zongming Ma and Xiangrui Meng.

lcda Latent Class Discriminant Analysis: local discrimination via latent class models. By Michael Buecker.

lmec Linear Mixed-Effects Models with Censored Responses. Fits a linear mixed-effects model in the formulation described in Laird and Ware (1982) but allowing for censored normal responses. In this version, the within group errors are assumed independent and identically distributed. By Florin Vaida and Lin Liu.

lmodel2 Computes model II simple linear regression using ordinary least squares (OLS), major axis (MA), standard major axis (SMA), and ranged major axis (RMA). By Pierre Legendre.

lmomRFA Functions for regional frequency analysis using the methods in "Regional frequency analysis: an approach based on L-moments" by J. R. M. Hosking and J. R. Wallis (1997). By J. R. M. Hosking.

localdepth Simplicial, Mahalanobis and ellipsoid local and global depth. By Claudio Agostinelli and Mario Romanazzi.

longRPart Recursive partitioning of longitudinal data using mixed-effects models. By Sam Stewart and Mohamed Abdolell.

mapReduce A flexible mapReduce framework for parallel computation. Provides (a) a pure R implementation, (b) a syntax following the mapReduce paper, and (c) a flexible and parallelizable back end. By Christopher Brown.

markerSearchPower Calculates statistical power of detecting associated markers based on one of the model selection strategies: marginal selection, exhaustive search, or forward selection. With assumed genetic effects (including interaction effect), allele frequencies, noise level, sample size, number of genotyped markers, and control level (i.e., number of markers/models intended to select), this package provides fast and accurate consultation on power of different model selection strategies. It helps researchers to decide a more efficient way for marker detection in genome-wide association studies. By Zheyang Wu and Hongyu Zhao.

mc2d Various distributions and utilities to build and study two-dimensional Monte Carlo simulations. By Régis Pouillot, Marie Laure Delignette-Muller and Jean-Baptiste Denis.

mcsm A collection of functions that allows the reenactment of the R programs used in the book "EnteR Monte Carlo Methods" without further programming. Programs can also be modified by the user to conduct one's own simulations. By Christian P. Robert.

medAdherence Medication Adherence: commonly used definitions. By Xiangyang Ye.

metaMA Meta-analysis for MicroArrays. Combines either p-values or modified effect sizes from different studies to find differentially expressed genes. By Guillemette Marot.

metacor Meta-analysis of correlation coefficients. Implement the DerSimonian-Laird (DSL) and Olkin-Pratt (OP) meta-analytical approaches with correlation coefficients as effect sizes. By Etienne Laliberté.

mhsmm Parameter estimation and prediction for hidden Markov and semi-Markov models for data with multiple observation sequences. The times must be equidistant but missing values of the observables are allowed. The observables are allowed to be multivariate. It is possible to have multiple sequences of data. Estimation of the parameters of the models are made using EM-algorithms. Computationally demanding parts of the algorithm are implemented in C for performance. Finally, the package is designed to allow users to specify their own emission distributions, giving flexibility in the type of data that may be analyzed. By Jared O'Connell and Søren Højsgaard.

minxent Implements entropy optimization distribution under specified constraints. Also offers an R interface to the MinxEnt and MaxEnt distributions. By Senay Asma.

mixAK Contains a mixture of statistical methods including MCMC methods to analyze normal mixtures. By Arnošt Komárek.

mixRasch Mixture Rasch Models with JMLE. Estimates mixture Rasch models, including the dichotomous Rasch model, the rating scale model, and the partial credit model. By John T. Willse.

mmcm Provides an implementation of the Modified Maximum Contrast Method. The current version features function `mmcm.resamp` which gives p-values for the modified maximum contrast statistics by using a resampling based procedure. By Kengo Nagashima and Yasunori Sato.

multicore Provides a way of running parallel computations in R on machines with multiple cores or CPUs, and methods for results collection. Jobs can share the entire initial workspace. By Simon Urbanek.

muscor MUlti-Stage COnvex Relaxation. Solves a number of convex/non-convex sparse regularization formulations for regression or binary classification using the algorithm described in "Multi-stage Convex Relaxation for Learning with Sparse Regularization". Loss functions include least squares, logistic, truncated ls/huber. By Tong Zhang.

mvgraph Multivariate interactive visualization. By Moritz Gschwandtner.

mvtBinaryEP Generates correlated binary data based on the method of Emrich and Piedmonte (1991). By Kunthel By and Bahjat Qaqish.

nleqslv Solve a system of non linear equations using a Broyden or a Newton method with a choice of global strategies such as linesearch and trust region. There are options for using a numerical or an analytical Jacobian and fixed or automatic scaling of parameters. By Berend Hasselman.

operators A set of operators for common tasks such as regex manipulation. By Romain François.

orthogonalsplinebasis Orthogonal *B*-spline basis functions. Represents the basis functions via a simple matrix formulation that facilitates taking integrals, derivatives, and orthogonalizing the basis functions. By Andrew Redd.

parcor Estimates the matrix of partial correlations based on different regularized regression methods: lasso, adaptive lasso, PLS, and Ridge Regression. By Nicole Krämer and Juliane Schäfer.

partDSA Partitioning using deletion, substitution, and addition moves, a novel tool for generating a piecewise constant estimation list of increasingly complex predictors based on an intensive and comprehensive search over the entire covariate space. By Annette Molinaro, Karen Lostrito, and Steve Weston.

pedigree Pedigree related functions. By Albart Coster.

penalizedSVM Feature selection SVM using penalty functions. The smoothly clipped absolute deviation (SCAD) and L_1 norm penalties are available up to now. By Natalia Becker, Wiebke Werft, and Axel Benner.

phmm Proportional Hazards Mixed-effects Model (PHMM). Fits PHMMs using an EM algorithm using Markov Chain Monte Carlo at the E-step. By Michael Donohue and Ronghui Xu.

plan Tools for project planning. Supports the creation of burndown charts. By Dan Kelley.

plotpc Plot principal component histograms around a bivariate scatter plot. By Stephen Milborrow.

plspm Partial Least Squares (PLS) methods with emphasis on structural equation models with latent variables. By Gaston Sanchez.

polydect Functions for one-dimension jump position detection using one-sided polynomial kernel detectors (polynomial order from 0 to 3). By Zhihua Su.

powerGWASinteraction Routines for power calculations for interactions for GWAS. By Charles Kooperberg.

powerSurvEpi Functions to calculate power and sample size for testing main effect or interaction effect in the survival analysis of epidemiological studies (non-randomized studies), taking into account the correlation between the covariate of the interest and other covariates. Some calculations also take into account the competing risks. Also includes functions to calculate power and sample size for testing main effect in the survival analysis of randomized clinical trials. By Weiliang Qiu, Jorge Chavarro, Ross Lazarus, Bernard Rosner, and Jing Ma.

prefmod Utilities to fit paired comparison models for preferences. Generates design matrix for analyzing real paired comparisons and derived paired comparison data (Likert type items, ratings or rankings) using a log-linear approach. Fits log-linear Bradley-Terry model (LLBT) exploiting an eliminate feature. Computes pattern models for paired comparisons, rankings, and ratings. Some treatment of missing values (MCAR and MNAR). By Reinhold Hatzinger.

primer Functions and data for the forthcoming book "A Primer of Ecology with R". Functions are primarily for systems of ordinary differential equations, difference equations, and eigenanalysis and projection of demographic matrices; data are for examples. By M. Henry H. Stevens.

pspearman Spearman's rank correlation test with improved accuracy. By Petr Savicky.

qtlbook Datasets for the book "A guide to QTL Mapping with R/qtl". By Karl W. Broman.

rSymPy Interface to access the SymPy computer algebra system running under Jython from R. By G. Grothendieck.

rbenchmark Benchmarking routine for R. Inspired by the Perl module Benchmark, and intended to facilitate benchmarking of arbitrary R code. Consists of just one function, `benchmark`, which is a simple wrapper around `system.time`. Given a specification of the benchmarking process (counts of replications, evaluation environment) and an arbitrary number of expressions, `benchmark` evaluates each of the expressions in the specified environment, replicating the evaluation as many times as specified, and returning the results conveniently wrapped into a data frame. By Wacek Kusnierczyk.

rcdklibs Provides the CDK libraries for use with R. To make use of the CDK within R, using the **rcdk** package, which exposes functionality in a more idiomatic way, is suggested; however, it is also possible to directly interact with CDK using **rJava**. By Rajarshi Guha.

rconifers Functions for simulating future forest conditions using the CONIFERS growth model. By Jeff D. Hamann and Martin W. Ritchie.

rela Item analysis with alpha standard error and principal axis factoring for continuous variable scales. By Michael Chajewski.

remMap Regularized Multivariate Regression for Identifying Master Predictors. Developed for fitting multivariate response regression models under the high-dimension-low-sample-size setting. By Jie Peng, Pei Wang, and Ji Zhu.

reporttools Generate LaTeX tables of descriptive statistics. Especially helpful when writing data analysis reports using Sweave. By Kaspar Rufibach.

rgrs Functions for beginners and social sciences students or researchers. Currently includes functions for cross-tabulation, weighting, results export, and maps plotting. Documentation and help pages are written in French. By Julien Barnier.

rngwell19937 Random number generator WELL19937a by F. Panneton, P. L'Ecuyer and M. Matsumoto with initialization using MRG32k5a by P. L'Ecuyer. WELL19937a is of similar type as Mersenne Twister and has the same period. It is slightly slower than Mersenne Twister, but has better equidistribution and "bit-mixing" properties and faster recovery from states with prevailing zeros than Mersenne Twister. The output function may be set to provide numbers with 53 or 32 random bits. By Petr Savicky.

robCompositions Methods for the imputation of compositional data including robust methods, (robust) outlier detection for compositional data, (robust) principal component analysis for compositional data, (robust) factor analysis for compositional data, and (robust) Anderson-Darling normality tests for compositional data. By Matthias Templ, Karel Hron, and Peter Filzmoser.

rsm Functions to generate response-surface designs, fit first- and second-order response-surface models, make contour plots, obtain the path of steepest ascent, and do canonical analysis. By Russell V. Lenth.

rtv Convenient representation and manipulation of realizations of Random Time Variables. By Charlotte Maia.

sabreR Statistical analysis of multi-process random effect response data by providing SABRE functionality from within R. Responses can take the form of binary, ordinal, count and linear recurrent events. Response sequences can be of different types. Such multi-process data is common in many research areas, e.g., the analysis of work and life histories. May be used for analyzing longitudinal data sets surveys either with recurrent information collected over time or with a clustered sampling scheme. By R. Crouchley.

scout Implements the Scout method for regression, described in "Covariance-regularized regression and classification for high-dimensional problems" by Witten and Tibshirani (2008, *Journal of the Royal Statistical Society, Series B*). By Daniela M. Witten and Robert Tibshirani.

sda Shrinkage Discriminant Analysis and feature selection. Provides an efficient framework for high-dimensional linear and diagonal discriminant analysis with feature selection. The classifier is trained using Stein-type shrinkage estimators and features are ranked using correlation-adjusted *t*-scores. Feature selection is controlled using false non-discovery rates or higher criticism scores. By Miika Ahdesmäki and Korbinian Strimmer.

sdcTable Statistical disclosure control for tabular data. By Bernhard Meindl.

sdtoolkit Scenario Discovery Tools to support robust decision making. Currently only implements a modified version of the Patient Rule Induction Method. By Benjamin P. Bryant.

selectiongain Calculates the gain from a model selection, using part of Tallis' (1961) algorithm. By Xuefei Mi, Xuefei Mi, and Friedrich Utz.

simone Statistical Inference for MOdular NEtworks (SIMoNe). Iteratively combines edge estimation and node classification on the basis of a mixture model for edge weight distributions. Edge inference may be managed via two alternative methods: GLasso and the Meinshausen-Bühlmann approach. Node Classification is managed by MixNet, a mixture model for random graphs. By Christophe Ambroise, Julien Chiquet, Gilles Grasseau, and Alexander Adam Smith.

sisus Stable Isotope Sourcing using Sampling: source partitioning using stable isotopes. By Erik Barry Erhardt.

sparseLDA Sparse linear discriminant analysis for gaussians and mixture of gaussians models. By Line Clemmensen, with contributions by Max Kuhn.

spatialsegregation Functionals and indices for measuring segregation in multitype spatial point patterns with graph based neighborhood description. Included indices: Mingling, Shannon, Simpson; functionals: Mingling, Shannon, Simpson, ISAR; neighborhoods: Geometric, *k*-nearest neighbors, Gabriel, Delauney. By Tuomas Rajala.

spcosa SPatial COverage SAmpling and random sampling from compact geographical strata created by *k*-means. By D. J. J. Walvoort, D. J. Brus, and J. J. de Gruijter.

spls Sparse Partial Least Squares (SPLS) regression. By Dongjun Chung, Hyonho Chun, and Sunduz Keles.

spuRs Functions and data sets from "An Introduction to Scientific Programming and Simulation, Using R". By Owen Jones, Robert Maillardet, Andrew Robinson, Olga Borovkova, and Steven Carnie.

survcomp Functions to assess and compare the performance of risk prediction (survival) models. By Benjamin Haibe-Kains, Christos Sotiriou, and Gianluca Bontempi.

tawny Provides various portfolio optimization strategies including random matrix theory and shrinkage estimators. Portfolio optimization typically requires an estimate of a covariance matrix of asset returns. There are many approaches for constructing such a covariance matrix, some using the sample covariance matrix as a starting point. This package provides implementations for two such methods: random matrix theory and shrinkage estimation. Each method attempts to clean or remove noise related to the sampling process from the sample covariance matrix. By Brian Lee Yung Rowe.

wasim Helpful tools for data processing and visualization of results of the hydrological model WASIM-ETH. By Dominik Reusser, Till Francke.

timeDate Chronological and calendarical objects for Rmetrics environment for teaching "Financial Engineering and Computational Finance". By Diethelm Würtz and Yohan Chalabi.

timeSeries Financial time series objects for the Rmetrics environment for teaching "Financial Engineering and Computational Finance". By Diethelm Würtz and Yohan Chalabi.

tlemix Trimmed maximum likelihood estimation: a general framework for robust fitting of finite mixture models. Parameter estimation is performed using the EM algorithm. By P. Neytchev, P. Filzmoser, R. Patnaik, A. Eisl and R. Boubela.

tmvtnorm Computes truncated multivariate normal probabilities, quantiles and densities, including one-dimensional marginal densities. By Stefan Wilhelm.

truncnorm Trunctated normal distribution. By Heike Trautmann, Detlef Steuer, and Olaf Mersmann.

tslars Least angle regression for time series analysis. By Sarah Gelper.

ucminf An algorithm for general-purpose unconstrained non-linear optimization. The algorithm is of quasi-Newton type with BFGS updating of the inverse Hessian and soft line search with a trust region type monitoring of the input to the line search algorithm. The interface of **ucminf** is designed for easy interchange with `optim`. By Hans Bruun Nielsen and Stig Bousgaard Mortensen.

uniCox Univarate shrinkage prediction for survival analysis using in the Cox model. Especially useful for high-dimensional data, including microarray data. By Rob Tibshirani.

vowels Manipulation, normalization, and plotting of phonetic and sociophonetic vowel formant data. Used as the backend for the NORM website. By Tyler Kendall and Erik R. Thomas.

vrmlgen Translates numerical data into 3-dimensional representations like 3D scatter plots, meshes, bar charts and graphs in the Virtual Reality Markup Language (VRML). By Enrico Glaab.

wasim Helpful tools for data processing and visualization of results of the hydrological model WASIM-ETH. By Dominik Reusser, Till Francke.

x12 A wrapper function and GUI for the X12 binaries under Windows. By Alexander Kowarik.

xterm256 Support for xterm256 escape sequences, enabling R functions to take advantage of foreground color, background color, in capable terminals. By Romain François.

yacca Yet Another Canonical Correlation Analysis package. Provides an alternative canonical correlation/redundancy analysis function, with associated print, plot, and summary methods. A method for generating helio plots is also included. By Carter T. Butts.

zyp Zhang and Yue-Pilon trends package. Contains an efficient implementation of Sen's slope method plus implementation of Xuebin Zhang's (Zhang, 1999) and Yue-Pilon's (Yue, 2002) prewhitening approaches to determining trends in climate data. By David Bronaugh and Arelia Werner for the Pacific Climate Impacts Consortium.

Other changes

- Package **Matrix** is recommended for R $\geq$ 2.9.0.
- Bundle **SciViews** was moved to the Archive and is replaced by its unbundled packages (**svGUI**, **svIDE**, **svMisc**, and **svSocket**).
- Packages **DDHFm**, **GeneTS**, **LLN**, **ProbForecastGOP**, **ProbeR**, **Rlab**, **Rmdr**, **Rmetrics**, **SparseLogReg**, **caretLSF**, **caretNWS**, **classPP**, **femmeR**, **gllm**, **knnFinder**, **intcox**, **mlica**, **mlmmm**, **mota**, **pARtial**, **rggm**, **sdtalt**, **survBayes**, and **waveclock** were moved to the Archive.
- Frontend **gnomeGUI** was moved to the Archive.

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

News from the Bioconductor Project

Bioconductor Team
Program in Computational Biology
Fred Hutchinson Cancer Research Center

We are pleased to announce Bioconductor 2.4, released on April 21, 2009. Bioconductor 2.4 is compatible with R 2.9.0, and consists of 320 packages. There are 28 new packages, and enhancements to many others. Explore Bioconductor at http://bioconductor.org, and install packages with

```
> source("http://bioconductor.org/biocLite.R")
> biocLite() # install standard packages...
> biocLite("IRanges") # ...or IRanges
```

New packages

This release includes powerful new packages for diverse areas of high-throughput analysis, including:

Refined differential expression power analysis, pre-processing and error estimation (**SSPA**, **dyebias**, **spkTools**, **Rmagpie**, **MCRestimate**).

Flow cytometry tools for data import (**flowflowJo**) and auto-gating (**flowStats**).

Advanced clustering and gene selection approaches (**Rmagpie**, **MCRestimate**, **GeneSelectMMD**, **tspair**, **metahdep**, **betr**).

Probabilistic graphical models for reverse engineering regulatory networks (**qpgraph**).

Pathway analysis using novel approaches (**KEGGgraph**, **geen2pathway**, **GOSemSim**, **SPIA**).

Technology-specific packages (**AffyTiling**, **rMAT**, **crlmm**, **GeneRegionScan**).

Interfaces to data base and other external resources (**biocDatasets**, **PAnnBuilder**, **DAVIDQuery**).

Annotation

Bioconductor 'annotation' packages contain biological information about microarray probes and the genes they are meant to interrogate, or contain ENTREZ gene based annotations of whole genomes. This release updates existing database content, and lays the groundwork for 4 new species: *Pan troglodytes*, *Macaca mulatta*, *Anopheles gambiae* and *Xenopus laevis*. These species will be available in the development branch starting in May. In addition, the 'yeast' package now contains NCBI identifiers. A similarly enhanced *Arabidopsis* package will be in the development branch in May.

High-throughput sequencing

The stable of tools for high-throughput sequence analysis has developed considerably during this release cycle, particularly data structures and methods for conveniently navigating this complex data. Examples include the `IRanges` and related classes and methods for manipulating ranged (interval-based) data, the `Rle` class and its rich functionality for run-length encoded data (e.g., genome-scale 'pileup' or coverage data), the `XDataFrame` class allowing data frame-like functionality but with more flexible column types (requiring only that the column object have methods for length and subsetting), and the `GenomeData` and `GenomeDataList` objects and methods for manipulating collections of structured (e.g., by chromosome or locus) data. The **Biostrings** package continues to provide very flexible pattern matching facilities, while **ShortRead** introduces new I/O functionality and the generation of HTML-based quality assessment reports from diverse data sources.

Other activities

Bioconductor package maintainers and the Bioconductor team invest considerable effort in producing high-quality software. A focus during development of Bioconductor 2.4 has been on more consistent and widespread use of name spaces and package imports. These changes reduce 'collisions' between user and package variable names, and make package code more robust. The Bioconductor team continues to ensure quality software through technical and scientific reviews of new packages, and daily builds of released packages on Linux, Windows, and Macintosh platforms. The Bioconductor web site is also evolving. Bioconductor 'views' describing software functionality have been re-organized, and package vignettes, reference manuals, and use statistics are readily accessible from package home pages.

Looking forward

The Bioconductor community will meet on July 27-28 at our annual conference in Seattle for a combination of scientific talks and hands-on tutorials. The active Bioconductor mailing lists (http://bioconductor.org/docs/mailList.html) connect users with each other, to domain experts, and to maintainers eager to ensure that their packages satisfy the needs of leading edge approaches.

This will be a dynamic release cycle. New contributed packages are already under review, and our

build machines have started tracking the latest development versions of R. In addition to development of high-quality algorithms to address microarray data analysis, we anticipate continued efforts to leverage diverse external data sources and to meet the challenges of presenting high volume data in rich graphical contexts.

R Foundation News

by Kurt Hornik

Donations and new members

Donations

Stefan Wyder, Switzerland

New supporting members

Stephan Devriese, Belgium
Frans van Dunné, Netherlands
Karl Ove Hufthammer, Norway
George Ostrouchov, USA
Kem Phillips, USA
Joon You, USA

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