R News

The Newsletter of the R Project

Volume 3/1, June 2003

Editorial

by Friedrich Leisch

Welcome to the first issue of R News in 2003, which is also the first issue where I serve as Editor-in-Chief. This year has already seen a lot of exciting activities in the R project, one of them was the Distributed Statistical Computing Workshop that took place here in Vienna. DSC 2003 is also partly responsible for the rather late release of this newsletter, both members of the editorial board and authors of regular columns were also involved in conference organization.

R 1.7.0 was released on 2003-04-16, see "Changes in R" for detailed release information. Two major new features of this release are support for name spaces in packages and stronger support of S4-style classes and methods. While the **methods** package by John Chambers provides R with S4 classes for some time now, R 1.7.0 is the first version of R where the **methods** package is attached by default every time R is started. I know of several users who were rather anxious about the possible side effects of R going further in the direction of S4, but the transitions seems to have gone very well (and we were not flooded by bug reports). Two articles by Luke Tierney and Doug Bates discuss name spaces and converting packages to S4, respectively.

In another article Jun Yan and Tony Rossini explain how Microsoft Windows versions of R and R packages can be cross-compiled on Linux machines.

Changing my hat from R News editor to CRAN maintainer, I want to take this opportunity to announce Uwe Ligges as new maintainer of the windows packages on CRAN. Until now these have been built by Brian Ripley, who has spent a lot of time and effort on the task. Thanks a lot, Brian!

Also in this issue, Greg Warnes introduces the **genetics** package, Jürgen Groß discusses the computation of variance inflation factors, Thomas Lumley shows how to analyze survey data in R, and Carson, Murison and Mason use parallel computing on a Beowulf cluster to speed up gene-shaving.

We also have a new column with book reviews. 2002 has seen the publication of several books explicitly written for R (or R and S-Plus) users. We hope to make book reviews a regular column of this newsletter. If you know of a book that should be reviewed, please tell the publisher to send a free review copy to a member of the editorial board (who will forward it to a referee).

Last, but not least, we feature a recreational page in this newsletter. Barry Rowlingson has written a crossword on R, and even offers a prize for the correct solution. In summary: We hope you will enjoy reading R News 3/1.

Friedrich Leisch
Technische Universität Wien, Austria
`Friedrich.Leisch@R-project.org`

Contents of this issue:

Name Space Management for R

Luke Tierney

Introduction

In recent years R has become a major platform for the development of new data analysis software. As the complexity of software developed in R and the number of available packages that might be used in conjunction increase, the potential for conflicts among definitions increases as well. The name space mechanism added in R 1.7.0 provides a framework for addressing this issue.

Name spaces allow package authors to control which definitions provided by a package are visible to a package user and which ones are private and only available for internal use. By default, definitions are private; a definition is made public by *exporting* the name of the defined variable. This allows package authors to create their main functions as compositions of smaller utility functions that can be independently developed and tested. Only the main functions are exported; the utility functions used in the implementation remain private. An incremental development strategy like this is often recommended for high level languages like R—a guideline often used is that a function that is too large to fit into a single editor window can probably be broken down into smaller units. This strategy has been hard to follow in developing R packages since it would have lead to packages containing many utility functions that complicate the user's view of the main functionality of a package, and that may conflict with definitions in other packages.

Name spaces also give the package author explicit control over the definitions of global variables used in package code. For example, a package that defines a function `mydnorm` as

```
mydnorm <- function(z)
    1/sqrt(2 * pi) * exp(- z^2 / 2)
```

most likely intends `exp`, `sqrt`, and `pi` to refer to the definitions provided by the **base** package. However, standard packages define their functions in the global environment shown in Figure 1.

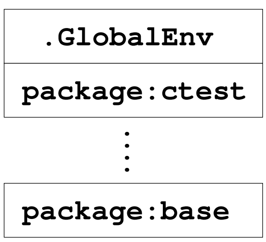


Figure 1: Dynamic global environment.

The global environment is dynamic in the sense that top level assignments and calls to `library` and `attach` can insert shadowing definitions ahead of the definitions in **base**. If the assignment `pi <- 3` has been made at top level or in an attached package, then the value returned by `mydnorm(0)` is not likely to be the value the author intended. Name spaces ensure that references to global variables in the **base** package cannot be shadowed by definitions in the dynamic global environment.

Some packages also build on functionality provided by other packages. For example, the package **modreg** uses the function `as.stepfun` from the **stepfun** package. The traditional approach for dealing with this is to use calls to `require` or `library` to add the additional packages to the search path. This has two disadvantages. First, it is again subject to shadowing. Second, the fact that the implementation of a package **A** uses package **B** does not mean that users who add **A** to their search path are interested in having **B** on their search path as well. Name spaces provide a mechanism for specifying package dependencies by *importing* exported variables from other packages. Formally importing variables form other packages again ensures that these cannot be shadowed by definitions in the dynamic global environment.

From a user's point of view, packages with a name space are much like any other package. A call to `library` is used to load the package and place its exported variables in the search path. If the package imports definitions from other packages, then these packages will be loaded but they will not be placed on the search path as a result of this load.

Adding a name space to a package

A package is given a name space by placing a 'NAMESPACE' file at the top level of the package source directory. This file contains *name space directives* that define the name space. This declarative approach to specifying name space information allows tools that collect package information to extract the package interface and dependencies without processing the package source code. Using a separate file also has the benefit of making it easy to add a name space to and remove a name space from a package.

The main name space directives are export and import directives. The `export` directive specifies names of variables to export. For example, the directive

```
export(as.stepfun, ecdf, is.stepfun, stepfun)
```

exports four variables. Quoting is optional for standard names such as these, but non-standard names need to be quoted, as in

```
export("[<-.tracker")
```

As a convenience, exports can also be specified with a pattern. The directive

```
exportPattern("\\.test$")
```

exports all variables ending with `.test`.

Import directives are used to import definitions from other packages with name spaces. The directive

```
import(mva)
```

imports all exported variables from the package **mva**. The directive

```
importFrom(stepfun, as.stepfun)
```

imports only the `as.stepfun` function from the **stepfun** package.

Two additional directives are available. The `S3method` directive is used to register methods for S3 generic functions; this directive is discussed further in the following section. The final directive is `useDynLib`. This directive allows a package to declaratively specify that a DLL is needed and should be loaded when the package is loaded.

For a package with a name space the two operations of loading the package and attaching the package to the search path are logically distinct. A package loaded by a direct call to `library` will first be loaded, if it is not already loaded, and then be attached to the search path. A package loaded to satisfy an import dependency will be loaded, but will *not* be attached to the search path. As a result, the single initialization hook function `.First.lib` is no longer adequate and is not used by packages with name spaces. Instead, a package with a name space can provide two hook functions, `.onLoad` and `.onAttach`. These functions, if defined, should not be exported. Many packages will not need either hook function, since import directives take the place of `require` calls and `useDynLib` directives can replace direct calls to `library.dynam`.

Name spaces are *sealed*. This means that , once a package with a name space is loaded, it is not possible to add or remove variables or to change the values of variables defined in a name space. Sealing serves two purposes: it simplifies the implementation, and it ensures that the locations of variable bindings cannot be changed at run time. This will facilitate the development of a byte code compiler for R (Tierney, 2001).

Adding a name space to a package may complicate debugging package code. The function `fix`, for example, is no longer useful for modifying an internal package function, since it places the new definition in the global environment and that new definition remains shadowed within the package by the original definition. As a result, it is a good idea not to add a name space to a package until it is completely debugged, and to remove the name space if further debugging is needed; this can be done by temporarily renaming the 'NAMESPACE' file. Other alternatives are being explored, and R 1.7.0 contains some experimental functions, such as `fixInNamespace`, that may help.

Name spaces and method dispatch

R supports two styles of object-oriented programming: the S3 style based on `UseMethod` dispatch, and the S4 style provided by the **methods** package. S3 style dispatch is still used the most and some support is provided in the name space implementation released in R 1.7.0.

S3 dispatch is based on concatenating the generic function name and the name of a class to form the name of a method. The environment in which the generic function is called is then searched for a method definition. With name spaces this creates a problem: if a package is imported but not attached, then the method definitions it provides may not be visible at the call site. The solution provided by the name space system is a mechanism for registering S3 methods with the generic function. A directive of the form

```
S3method(print, dendrogram)
```

in the 'NAMESPACE' file of a package registers the function `print.dendrogram` defined in the package as the `print` method for class `dendrogram`. The variable `print.dendrogram` does not need to be exported. Keeping the method definition private ensures that users will not inadvertently call the method directly.

S4 classes and generic functions are by design more formally structured than their S3 counterparts and should therefore be conceptually easier to integrate with name spaces than S3 generic functions and classes. For example, it should be fairly easy to allow for both exported and private S4 classes; the concatenation-based approach of S3 dispatch does not seem to be compatible with having private classes except by some sort of naming convention. However, the integration of name spaces with S4 dispatch could not be completed in time for the release of R 1.7.0. It is quite likely that the integration can be accomplished by adding only two new directives, `exportClass` and `importClassFrom`.

A simple example

A simple artificial example may help to illustrate how the import and export directives work. Consider two packages named **foo** and **bar**. The R code for package **foo** in file 'foo.R' is

```
x <- 1
f <- function(y) c(x,y)
```

The 'NAMESPACE' file contains the single directive

```
export(f)
```

Thus the variable `x` is private to the package and the function `f` is public. The body of `f` references a global variable `x` and a global function `c`. The global refrence `x` corresponds to the definition in the package. Since the package does not provide a definition for `c` and does not import any definitions, the variable `c` refers to the definition provided by the **base** package.

The second package **bar** has code file 'bar.R' containing the definitions

```
c <- function(...) sum(...)
g <- function(y) f(c(y, 7))
```

and 'NAMESPACE' file

```
import(foo)
export(g)
```

The function `c` is private to the package. The function `g` calls a global function `c`. Definitions provided in the package take precedence over imports and definitions in the **base** package; therefore the definition of `c` used by `g` is the one provided in **bar**.

Calling `library(bar)` loads **bar** and attaches its exports to the search path. Package **foo** is also loaded but not attached to the search path. A call to `g` produces

```
> g(6)
[1]  1 13
```

This is consistent with the definitions of `c` in the two settings: in **bar** the function `c` is defined to be equivalent to `sum`, but in **foo** the variable `c` refers to the standard function `c` in **base**.

Some details

This section provides some details on the current implementation of name spaces. These details may change as name space support evolves.

Name spaces use R environments to provide static binding of global variables for function definitions. In a package with a name space, functions defined in the package are defined in a name space environment as shown in Figure 2. This environment consists of a set of three static frames followed by the usual dynamic global environment. The first static frame contains the local definitions of the package. The second frame contains imported definitions, and the third frame is the **base** package. Suppose the function `mydnorm` shown in the introduction is defined in a package with a name space that does not explicitly import any definitions. Then a call to `mydnorm` will cause R to evaluate the variable `pi` by searching for a binding in the name space environment. Unless the package itself defines a variable `pi`, the first binding found will be the one in the third frame, the definition provided in the **base** package. Definitions in the dynamic global environment cannot shadow the definition in **base**.

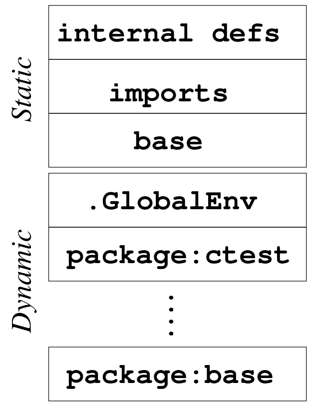


Figure 2: Name space environment.

When `library` is called to load a package with a name space, `library` first calls `loadNamespace` and then `attachNamespace`. `loadNamespace` first checks whether the name space is already loaded and registered with the internal registry of loaded name spaces. If so, the already loaded name space is returned. If not, then `loadNamespace` is called on all imported name spaces, and definitions of exported variables of these packages are copied to the imports frame, the second frame, of the new name space. Then the package code is loaded and run, and exports, S3 methods, and shared libraries are registered. Finally, the `.onLoad` hook is run, if the package defines one, and the name space is sealed. Figure 3 illustrates the dynamic global environment and the internal data base of loaded name spaces after two packages, **A** and **B**, have been loaded by explicit calls to `library` and package **C** has been loaded to satisfy import directives in package **B**.

The serialization mechanism used to save R workspaces handles name spaces by writing out a reference for a name space consisting of the package name and version. When the workspace is loaded, the reference is used to construct a call to `loadNamespace`. The current implementation ignores the version information; in the future this may be used to signal compatibility warnings or select among several available versions.

The registry of loaded name spaces can be examined using `loadedNamespaces`. In the current implementation loaded name spaces are not unloaded automatically. Detaching a package with a

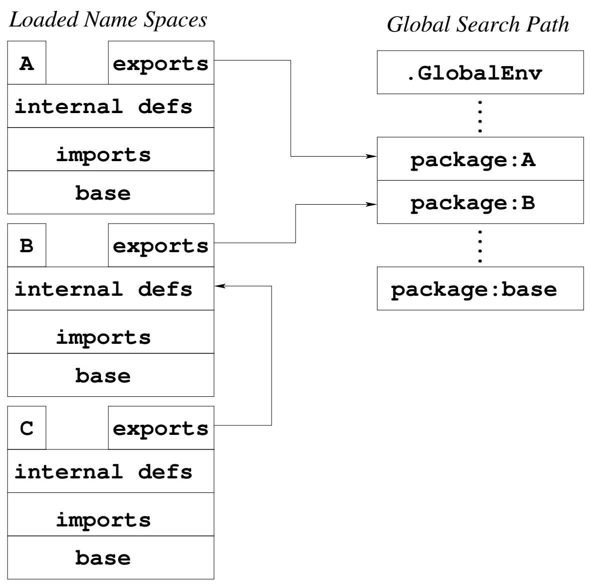


Figure 3: Internal implementation.

name space loaded by a call to `library` removes the frame containing the exported variables of the package from the global search path, but it does not unload the name space. Future implementations may automatically unload name spaces that are no longer needed to satisfy import dependencies. The function `unloadNamespace` can be used to force unloading of a name space; this can be useful if a new version of the package is to be loaded during development.

Variables exported by a package with a name space can also be referenced using a fully qualified variable reference consisting of the package name and the variable name separated by a double colon, such as `foo::f`. This can be useful in debugging at the top level. It can also occasionally be useful in source code, for example in cases where functionality is only to be used if the package providing it is available. However, dependencies resulting from use of fully qualified references are not visible from the name space directives and therefore may cause some confusion, so explicit importing is usually preferable. Computing the value of a fully qualified variable reference will cause the specified package to be loaded if it is not loaded already.

Discussion

Name spaces provide a means of making R packages more modular. Many languages and systems originally developed without name space or module systems have had some form of name space management added in order to support the development of larger software projects. C++ and Tcl (Welch, 1999) are two examples. Other languages, such as Ada (Barnes, 1998) and Modula-3 (Harbison, 1992) were designed from the start to include powerful module systems. These languages use a range of approaches for specifying name space or module information. The declarative approach with separation of implementation code and name space information used in the system included in R 1.7.0 is closest in spirit to the approaches of Modula-3 and Ada. In contrast, Tcl and C++ interleave name space specification and source code.

The current R name space implementation, while already very functional and useful, is still experimental and incomplete. Support for S4 methods still needs to be included. More exhaustive error checking is needed, along with code analysis tools to aid in constructing name space specifications. To aid in package development and debugging it would also be useful to explore whether strict sealing can be relaxed somewhat without complicating the implementation. Early experiments suggest that this may be possible.

Some languages, including Ada, ML (Ullman, 1997) and the MzScheme Scheme implementation (PLT) provide a richer module structure in which parameterized partial modules can be assembled into complete modules. In the context of R this might correspond to having a parameterized generic data base access package that is completed by providing a specific data base interface. Whether this additional power is useful and the associated additional complexity is warranted in R is not yet clear.

Bibliography

John Barnes. *Programming In Ada 95*. Addison-Wesley, 2nd edition, 1998. 5

Samuel P. Harbison. *Modula-3*. Prentice Hall, 1992. 5

PLT. PLT MzScheme. World Wide Web, 2003. URL http://www.plt-scheme.org/software/mzscheme/. 5

Luke Tierney. Compiling R: A preliminary report. In Kurt Hornik and Friedrich Leisch, editors, *Proceedings of the 2nd International Workshop on Distributed Statistical Computing, March 15-17, 2001, Technische Universität Wien, Vienna, Austria*, 2001. URL http://www.ci.tuwien.ac.at/Conferences/DSC-2001/Proceedings/. ISSN 1609-395X. 3

Jeffrey Ullman. *Elements of ML Programming*. Prentice Hall, 1997. 5

Brent Welch. *Practical Programming in Tcl and Tk*. Prentice Hall, 1999. 5

Luke Tierney
Department of Statistics & Actuarial Science
University of Iowa, Iowa City, Iowa, U.S.A
luke@stat.uiowa.edu

Converting Packages to S4

by Douglas Bates

Introduction

R now has two systems of classes and methods, known informally as the 'S3' and 'S4' systems. Both systems are based on the assignment of a *class* to an object and the use of *generic functions* that invoke different *methods* according to the class of their arguments. Classes organize the representation of information and methods organize the actions that are applied to these representations.

'S3' classes and methods for the S language were introduced in Chambers and Hastie (1992), (see also Venables and Ripley, 2000, ch. 4) and have been implemented in R from its earliest public versions. Because many widely-used R functions, such as `print`, `plot` and `summary`, are S3 generics, anyone using R inevitably (although perhaps unknowingly) uses the S3 system of classes and methods.

Authors of R packages can take advantage of the S3 class system by assigning a class to the objects created in their package and defining methods for this class and either their own generic functions or generic functions defined elsewhere, especially those in the `base` package. Many R packages do exactly this. To get an idea of how many S3 classes are used in R, attach the packages that you commonly use and call `methods("print")`. The result is a vector of names of methods for the `print` generic. It will probably list dozens, if not hundreds, of methods. (Even that list may be incomplete because recent versions of some popular packages use namespaces to hide some of the S3 methods defined in the package.)

The S3 system of classes and methods is both popular and successful. However, some aspects of its design are at best inconvenient and at worst dangerous. To address these inadequacies John Chambers introduced what is now called the 'S4' system of classes and methods (Chambers, 1998; Venables and Ripley, 2000). John implemented this system for R in the `methods` package which, beginning with the 1.7.0 release of R, is one of the packages that are attached by default in each R session.

There are many more packages that use S3 classes and methods than use S4. Probably the greatest use of S4 at present is in packages from the Bioconductor project (Gentleman and Carey, 2002). I hope by this article to encourage package authors to make greater use of S4.

Conversion from S3 to S4 is not automatic. A package author must perform this conversion manually and the conversion can require a considerable amount of effort. I speak from experience. Saikat DebRoy and I have been redesigning the linear mixed-effects models part of the `nlme` package, including conversion to S4, and I can attest that it has been a lot of work. The purpose of this article is to indicate what is gained by conversion to S4 and to describe some of our experiences in doing so.

S3 versus S4 classes and methods

S3 classes are informal: the class of an object is determined by its class attribute, which should consist of one or more character strings, and methods are found by combining the name of the generic function with the class of the first argument to the function. If a function having this combined name is on the search path, it is assumed to be the appropriate method. Classes and their contents are not formally defined in the S3 system - at best there is a "gentleman's agreement" that objects in a class will have certain structure with certain component names.

The informality of S3 classes and methods is convenient but dangerous. There are obvious dangers in that any R object can be assigned a class, say `"foo"`, without any attempt to validate the names and types of components in the object. That is, there is no guarantee that an object that claims to have class `"foo"` is compatible with methods for that class. Also, a method is recognized solely by its name so a function named `print.foo` is assumed to be the method for the `print` generic applied to an object of class `foo`. This can lead to surprising and unpleasant errors for the unwary.

Another disadvantage of using function names to identify methods is that the class of only one argument, the first argument, is used to determine the method that the generic will use. Often when creating a plot or when fitting a statistical model we want to examine the class of more than one argument to determine the appropriate method, but S3 does not allow this.

There are more subtle disadvantages to the S3 system. Often it is convenient to describe a class as being a special case of another class; for example, a model fit by `aov` is a special case of a linear model (class `lm`). We say that class `aov` inherits from class `lm`. In the informal S3 system this inheritance is described by assigning the class `c("aov","lm")` to an object fit by `aov`. Thus the inheritance of classes becomes a property of the object, not a property of the class, and there is no guarantee that it will be consistent across all members of the class. Indeed there were examples in some packages where the class inheritance was not consistently assigned.

By contrast, S4 classes must be defined explicitly. The number of slots in objects of the class, and the names and classes of the slots, are established at the time of class definition. When an object of the class is created, and at some points during computation with the object, it is validated against the definition. Inheritance of classes is also specified at the time of class definition and thus becomes a property of the class, not a (possibly inconsistent) property of objects in the class.

S4 also requires formal declarations of methods, unlike the informal system of using function names to identify a method in S3. An S4 method is declared by a call to `setMethod` giving the name of the generic and the "signature" of the arguments. The signature identifies the classes of one or more named arguments to the generic function. Special meta-classes named `ANY` and `missing` can be used in the signature.

S4 generic functions are automatically created when a method is declared for an existing function, in which case the function becomes generic and the current definition becomes the default method. A new generic function can be declared explicitly by a call to `setGeneric`. When a method for the generic is declared with `setMethod` the number, names, and order of its arguments are matched against those of the generic. (If the generic has a `...` argument, the method can add new arguments but it can never omit or rename arguments of the generic.)

In summary, the S4 system is much more formal regarding classes, generics, and methods than is the S3 system. This formality means that more care must be taken in the design of classes and methods for S4. In return, S4 provides greater security, a more well-defined organization, and, in my opinion, a cleaner structure to the package.

Package conversion

Chambers (1998, ch. 7,8) and Venables and Ripley (2000, ch. 5) discuss creating new packages based on S4. Here I will concentrate on converting an existing package from S3 to S4.

S4 requires formal definitions of generics, classes, and methods. The generic functions are usually the easiest to convert. If the generic is defined externally to the package then the package author can simply begin defining methods for the generic, taking care to ensure that the argument sequences of the method are compatible with those of the generic. As described above, assigning an S4 method for, say, `coef` automatically creates an S4 generic for `coef` with the current externally-defined `coef` function as the default method. If an S3 generic was defined internally to the package then it is easy to convert it to an S4 generic.

Converting classes is less easy. We found that we could use the informal set of classes from the S3 version as a guide when formulating the S4 classes but we frequently reconsidered the structure of the classes during the conversion. We frequently found ourselves adding slots during the conversion so we had more slots in the S4 classes than components in the S3 objects.

Increasing the number of slots may be an inevitable consequence of revising the package (we tend to add capabilities more frequently than we remove them) but it may also be related to the fact that S4 classes must be declared explicitly and hence we must consider the components or slots of the classes and the relationships between the classes more carefully.

Another reason that we incorporating more slots in the S4 classes than in the S3 prototype is because we found it convenient that the contents of slots in S4 objects can easily be accessed and modified in C code called through the `.Call` interface. Several C macros for working with S4 classed objects, including `GET_SLOT`, `SET_SLOT`, `MAKE_CLASS` and `NEW` (all described in Chambers (1998)) are available in R. The combination of the formal classes of S4, the `.Call` interface, and these macros allows a programmer to manipulate S4 classed objects in C code nearly as easily as in R code. Furthermore, when the C code is called from a method, the programmer can be confident of the classes of the objects passed in the call and the classes of the slots of those objects. Much of the checking of classes or modes and possible coercion of modes that is common in C code called from R can be bypassed.

We found that we would initially write methods in R then translate them into C if warranted. The nature of our calculations, frequently involving multiple decompositions and manipulations of sections of arrays, was such that the calculations could be expressed in R but not very cleanly. Once we had the R version working satisfactorily we could translate into C the parts that were critical for performance or were awkward to write in R. An important advantage of this mode of development is that we could use the same slots in the C version as in the R version and create the same types of objects to be returned.

We feel that defining S4 classes and methods in R then translating parts of method definitions to C functions called through `.Call` is an extremely effective mode for numerical computation. Programmers who have experience working in C++ or Java may initially find it more convenient to define classes and methods in the compiled language and perhaps define a parallel series of classes in R. (We did exactly that when creating the Matrix package for R.) We encourage such programmers to try instead this method of defining only one set of classes, the S4 classes in R, and use these classes in both the interpreted language and the compiled language.

The definition of S4 methods is more formal than in S3 but also more flexible because of the ability to match an argument signature rather than being contrained to matching just the class of the first argument. We found that we used this extensively when defining methods for generics that fit models. We would define the "canonical" form of the arguments, which frequently was a rather wordy form, and one "collector" method that matched this form of the arguments. The collector method is the one that actually does the work, such as fitting the model. All the other methods are designed to take more conveniently expressed forms of the arguments and rearrange them into the canonical form. Our subjective impression is that the resulting methods are much easier to understand than those in the S3 version.

Pitfalls

S4 classes and methods are powerful tools. With these tools a programmer can exercise fine-grained control over the definition of classes and methods. This encourages a programming style where many specialized classes and methods are defined. One of the difficulties that authors of packages then face is documenting all these classes, generics, and methods.

We expect that generic functions and classes will be described in detail in the documentation, usually in a separate documentation file for each class and each generic function, although in some cases closely related classes could be described in a single file. It is less obvious whether, how, and where to document methods. In the S4 system methods are associated with a generic function and an argument signature. It is not clear if a given method should be documented separately or in conjunction with the generic function or with a class definition.

Any of these places could make sense for some methods. All of them have disadvantages. If all methods are documented separately there will be an explosion of the number of documentation files to create and maintain. That is an unwelcome burden. Documenting methods in conjunction with the generic can work for internally defined generics but not for those defined externally to the package. Documenting methods in conjunction with a class only works well if the method signature consists of one argument only but part of the power of S4 is the ability to dispatch on the signature of multiple arguments.

Others working with S4 packages, including the Bioconductor contributors, are faced with this problem of organizing documentation. Gordon Smyth and Vince Carey have suggested some strategies for organizing documentation but more experience is definitely needed.

One problem with creating many small methods that rearrange arguments and then call `callGeneric()` to get eventually to some collector method is that the sequence of calls has to be unwound when returning the value of the call. In the case of a model-fitting generic it is customary to preserve the original call in a slot or component named `call` so that it can be displayed in summaries and also so it can be used to `update` the fitted model. To preserve the call, each of the small methods that just manipulate the arguments must take the result of `callGeneric`, reassign the `call` slot and return this object. We discovered, to our chagrin, that this caused the entire object, which can be very large in some of our examples, to be copied in its entirety as each method call was unwound.

Conclusions

Although the formal structure of the S4 system of classes and methods requires greater discipline by the programmer than does the informal S3 system, the resulting clarity and security of the code makes S4 worthwhile. Moreover, the ability in S4 to work with a single class structure in R code and in C code to be called by R is a big win.

S4 encourages creating many related classes and many methods for generics. Presently this creates difficult decisions on how to organize documentation and how unwind nested method calls without unwanted copying of large objects. However it is still early days with regard to the construction of large packages based on S4 and as more experience is gained we will expect that knowledge of the best practices will be disseminated in the community so we can all benefit from the S4 system.

Bibliography

J. M. Chambers. *Programming with Data*. Springer, New York, 1998. ISBN 0-387-98503-4. 6, 7

J. M. Chambers and T. J. Hastie. *Statistical Models in S*. Chapman & Hall, London, 1992. ISBN 0-412-83040-X. 6

R. Gentleman and V. Carey. Bioconductor. *R News*, 2(1):11–16, March 2002. URL http://CRAN.R-project.org/doc/Rnews/. 6

W. N. Venables and B. D. Ripley. *S Programming*. Springer, 2000. URL http://www.stats.ox.ac.uk/pub/MASS3/Sprog/. ISBN 0-387-98966-8. 6, 7

Douglas Bates
University of Wisconsin–Madison, U.S.A.
Bates@stat.wisc.edu

The genetics Package

Utilities for handling genetic data

by Gregory R. Warnes

Introduction

In my work as a statistician in the Non-Clinical Statistics and Biostatistical Applications group within Pfizer Global Research and Development I have the opportunity to perform statistical analysis in a wide variety of domains. One of these domains is pharmacogenomics, in which we attempt to determine the relationship between the genetic variability of individual patients and disease status, disease progression, treatment efficacy, or treatment side effect profile.

Our normal approach to pharmacogenomics is to start with a small set of candidate genes. We then look for markers of genetic variability within these genes. The most common marker types we encounter are Single Nucleotide Polymorphisms (SNPs). SNPs are locations where some individuals differ from the norm by the substitution one of the 4 DNA bases, adenine (A), thymine (T), guanine (G), and cytosine (C), by one of the other bases. For example, a single cytosine (C) might be replaced by a single tyrosine (T) in the sequence 'CCT**C**AGC', yielding 'CCT**T**AGC'. We also encounter simple sequence length polymorphisms (SSLP), which are also known as microsatellite DNA. SSLP are simple reteating patters of bases where the number of repeats can vary. E.g., at a particular position, some individuals might have 3 repeats of the pattern 'CT', 'AC**CTCTCT**AA', while others might have 5 repeats, 'AC**CTCTCTCTCT**AA'.

Regardless of the type or location of genetic variation, each individual has two copies of each chromosome, hence two alleles (variants), and consequently two data values for each marker. This information is often presented together by providing a pair of allele names. Sometimes a separator is used (e.g. 'A/T'), sometimes they are simply concatenated (e.g., 'AT').

A further layer of complexity arises from the inability of most laboratory methods to determine which observed variants comes from which copy of the chromosome. (Statistical methods are often necessary to impute this information when it is needed.) For this type of data 'A/T', and 'T/A' are equivalent.

The genetics package

The **genetics** package, available from CRAN, includes classes and methods for creating, representing, and manipulating genotypes (unordered allele pairs) and haplotypes (ordered allele pairs). Genotypes and haplotypes can be annotated with chromosome, locus (location on a chromosome), gene, and marker information. Utility functions compute genotype and allele frequencies, flag homozygotes or heterozygotes, flag carriers of certain alleles, count the number of a specific allele carried by an individual, extract one or both alleles. These functions make it easy to create and use single-locus genetic information in R's statistical modeling functions.

The **genetics** package also provides a set of functions to estimate and test for departure from Hardy-Weinberg equilibrium (HWE). HWE specifies the expected allele frequencies for a single population when none of the variant alleles impart a survival benefit. Departure from HWE is often indicative of a problem with the laboratory assay, and is often the first statistical method applied to genetic data. In addition, the **genetics** package provides functions to test for linkage disequilibrium (LD), the non-random association of marker alleles which can arise from marker proximity or from selection bias. Further, to assist in sample size calculations when considering sample sizes needed when investigating potential markers, we provide a function which computes the probability of observing all alleles with a given true frequency.

My primary motivation in creating the **genetics** package was to overcome the difficulty in representing and manipulating genotype in general-purpose statistical packages. Without an explicit genotype variable type, handling genetic variables requires considerable string manipulation, which can be quite messy and tedious. The `genotype` function has been designed to remove the need to perform string manupulation by allowing allele pairs to be specified in any of four commonly occuring notations:

- A single vector with a character separator:

```
g1 <- genotype( c('A/A','A/C','C/C','C/A',
                  NA,'A/A','A/C','A/C') )
g3 <- genotype( c('A A','A C','C C','C A',
                  '','A A','A C','A C'),
                sep=' ', remove.spaces=F)
```

- A single vector with a positional separator

```
g2 <- genotype( c('AA','AC','CC','CA','',
                  'AA','AC','AC'), sep=1 )
```

- Two separate vectors

```
g4 <- genotype(
        c('A','A','C','C','','A','A','A'),
        c('A','C','C','A','','A','C','C')
        )
```

- A dataframe or matrix with two columns

```
gm <- cbind(
        c('A','A','C','C','','A','A','A'),
        c('A','C','C','A','','A','C','C') )
g5 <- genotype( gm )
```

For simplicity, the functions makeGenotype and makeHaplotype can be used to convert all of the genetic variables contained in a dataframe in a single pass. (See the help page for details.)

A second difficulty in using genotypes is the need to represent the information in different forms at different times. To simplify the use of genotype variables, each of the three basic ways of modeling the effect of the allele combinations is directly supported by the **genetics** package:

categorical Each allele combination acts differently.

This situation is handled by entering the `genotype` variable without modification into a model. In this case, it will be treated as a factor:

```
lm( outcome ~ genotype.var + confounder )
```

additive The effect depends on the number of copies of a specific allele (0, 1, or 2).

The function `allele.count( gene, allele )` returns the number of copies of the specified allele:

```
lm( outcome ~ allele.count(genotype.var,'A')
              + confounder )
```

dominant/recessive The effect depends only on the presence or absence of a specific allele.

The function `carrier( gene, allele )` returns a boolean flag if the specified allele is present:

```
lm( outcome ~ carrier(genotype.var,'A')
              + confounder )
```

Implementation

The basic functionality of the **genetics** package is provided by the `genotype` class and the `haplotype` class, which is a simple extension of the former. Friedrich Leisch and I collaborated on the design of the `genotype` class. We had four goals: First, we wanted to be able to manipulate both alleles as a single variable. Second, we needed a clean way of accessing the individual alleles when this was required. Third, a genotype variable should be able to be stored in dataframes as they are currently implemented in R. Fourth, the implementation of genotype variables should be space-efficient.

After considering several potential implementations, we chose to implement the genotype class as an extension to the in-built factor variable type with additional information stored in attributes. Genotype objects are stored as factors and have the class list `c("genotype","factor")`. The names of the factor levels are constructed as `paste(allele1,"/",allele2,sep="")`. Since most genotyping methods do not indicate which allele comes from which member of a chromosome pair, the alleles for each individual are placed in a consistent order controlled by the `reorder` argument. In cases when the allele order is informative, the `haplotype` class, which preserves the allele order, should be used instead.

The set of allele names is stored in the attribute `allele.names`. A translation table from the factor levels to the names of each of the two alleles is stored in the attribute `allele.map`. This map is a two column character matrix with one row per factor level. The columns provide the individual alleles for each factor level. Accesing the individual alleles, as performed by the `allele` function, is accomplished by simply indexing into this table,

```
allele.x <- attrib(x,"allele.map")
alleles.x[genotype.var,which]
```

where `which` is `1`, `2`, or `c(1,2)` as appropriate.

Finally, there is often additional meta-information associated with a genotype. The functions `locus`, `gene`, and `marker` create objects to store information, respectively, about genetic loci, genes, and markers. Any of these objects can be included as part of a genotype object using the `locus` argument. The print and summary functions for genotype objects properly display this information when it is present.

This implementation of the genotype class met our four design goals and offered an additional benefit: in most contexts factors behave the same as the desired default behavior for genotype objects. Consequently, relatively few additional methods needed to written. Further, in the absence of the **genetics** package, the information stored in genotype objects is still accessible in a reasonable way.

The `genotype` class is accompanied by a full complement of helper methods for standard R operators (`[]`, `[<-`, `==`, etc.) and object methods (`summary`, `print`, `is.genotype`, `as.genotype`, etc.). Additional functions for manipulating genotypes include:

allele Extracts individual alleles. matrix.

allele.names Extracts the set of allele names.

homozygote Creates a logical vector indicating whether both alleles of each observation are the same.

heterozygote Creates a logical vector indicating whether the alleles of each observation differ.

carrier Creates a logical vector indicating whether the specified alleles are present.

allele.count Returns the number of copies of the specified alleles carried by each observation.

getlocus Extracts locus, gene, or marker information.

makeGenotypes Convert appropriate columns in a dataframe to genotypes or haplotypes

write.pop.file Creates a 'pop' data file, as used by the GenePop (http://wbiomed.curtin.edu.au/genepop/) and LinkDos (http://wbiomed.curtin.edu.au/genepop/linkdos.html) software packages.

write.pedigree.file Creates a 'pedigree' data file, as used by the QTDT software package (http://www.sph.umich.edu/statgen/abecasis/QTDT/).

write.marker.file Creates a 'marker' data file, as used by the QTDT software package (http://www.sph.umich.edu/statgen/abecasis/QTDT/).

The **genetics** package provides four functions related to Hardy-Weinberg Equilibrium:

diseq Estimate or compute confidence interval for the single marker Hardy-Weinberg disequilibrium

HWE.chisq Performs a Chi-square test for Hardy-Weinberg equilibrium

HWE.exact Performs a Fisher's exact test of Hardy-Weinberg equilibrium for two-allele markers.

HWE.test Computes estimates and bootstrap confidence intervals, as well as testing for Hardy-Weinberg equilibrium.

as well as three related to linkage disequilibrium (LD):

LD Compute pairwise linkage disequilibrium between genetic markers.

LDtable Generate a graphical table showing the LD estimate, number of observations and p-value for each marker combination, color coded by significance.

LDplot Plot linkage disequilibrium as a function of marker location.

and one function for sample size calculation:

gregorius Probability of Observing All Alleles with a Given Frequency in a Sample of a Specified Size.

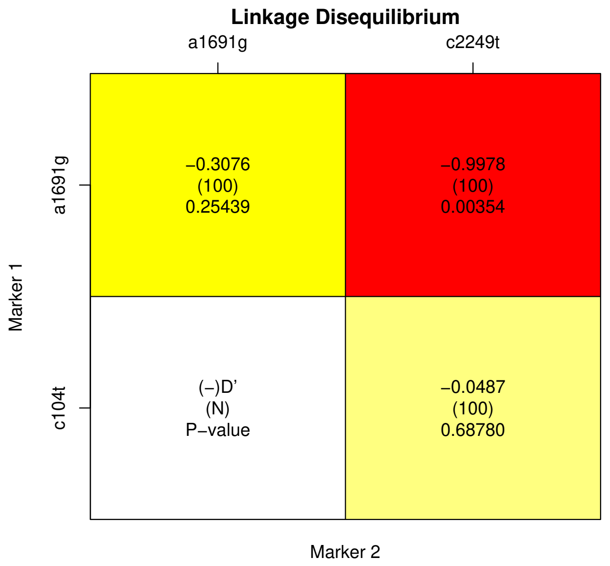
The algorithms used in the HWE and LD functions are beyond the scope of this article, but details are provided in the help pages or the corresponding package documentation.

Example

Here is a partial session using tools from the genotype package to examine the features of 3 simulated markers and their relationships with a continuous outcome:

```
> library(genetics)
[...]
> # Load the data from a CSV file
> data <- read.csv("example_data.csv")
>
> # Convert genotype columns to genotype variables
> data <- makeGenotypes(data)
>
> ## Annotate the genes
> marker(data$a1691g) <-
+   marker(name="A1691G",
+          type="SNP",
+          locus.name="MBP2",
+          chromosome=9,
+          arm="q",
+          index.start=35,
+          bp.start=1691,
+          relative.to="intron 1")
[...]
>
> # Look at some of the data
> data[1:5,]
      PID DELTA.BMI c104t a1691g c2249t
1 1127409      0.62   C/C    G/G    T/T
2  246311      1.31   C/C    A/A    T/T
3  295185      0.15   C/C    G/G    T/T
4   34301      0.72   C/T    A/A    T/T
5   96890      0.37   C/C    A/A    T/T
>
> # Get allele information for c104t
> summary(data$c104t)

Marker: MBP2:C-104T (9q35:-104) Type: SNP

Allele Frequency:
  Count Proportion
C   137       0.68
T    63       0.32

Genotype Frequency:
    Count Proportion
C/C    59       0.59
C/T    19       0.19
T/T    22       0.22


>
>
> # Check Hardy-Weinberg Equilibrium
> HWE.test(data$c104t)

        -----------------------------------
        Test for Hardy-Weinberg-Equilibrium
        -----------------------------------

Call:
```

```
HWE.test.genotype(x = data$c104t)

Raw Disequlibrium for each allele pair (D)

  C    T
C      0.12
T 0.12

Scaled Disequlibrium for each allele pair (D')

  C    T
C      0.56
T 0.56

Correlation coefficient for each allele pair (r)

  C     T
C  1.00 -0.56
T -0.56  1.00

Overall Values

     Value
  D   0.12
  D'  0.56
  r  -0.56

Confidence intervals computed via bootstrap
using 1000 samples

             Observed 95% CI           NA's
  Overall D   0.121   ( 0.073,  0.159) 0
  Overall D'  0.560   ( 0.373,  0.714) 0
  Overall r  -0.560   (-0.714, -0.373) 0
             Contains Zero?
  Overall D  *NO*
  Overall D' *NO*
  Overall r  *NO*

Significance Test:

        Exact Test for Hardy-Weinberg Equilibrium

data:  data$c104t
N11 = 59, N12 = 19, N22 = 22, N1 = 137, N2
= 63, p-value = 3.463e-08



>
> # Check Linkage Disequilibrium
> ld <- LD(data)
Warning message:
Non-genotype variables or genotype variables with
more or less than two alleles detected. These
 variables will be omitted: PID, DELTA.BMI
in: LD.data.frame(data)
> ld # text display

Pairwise LD
-----------
               a1691g c2249t
c104t  D        -0.01  -0.03
c104t  D'        0.05   1.00
c104t  Corr.    -0.03  -0.21
c104t  X^2       0.16   8.51
c104t  P-value   0.69 0.0035
c104t  n          100    100

a1691g D               -0.01
a1691g D'               0.31
a1691g Corr.           -0.08
a1691g X^2              1.30
a1691g P-value          0.25
a1691g n                 100


>
> LDtable(ld) # graphical display
```




```
> # fit a model
> summary(lm( DELTA.BMI ~
+                homozygote(c104t,'C') +
+                allele.count(a1691g, 'G') +
+                c2249t, data=data))

Call:
lm(formula = DELTA.BMI ~ homozygote(c104t, "C") +
    allele.count(a1691g, "G") + c2249t,
    data = data)

Residuals:
    Min      1Q  Median      3Q     Max
-2.9818 -0.5917 -0.0303  0.6666  2.7101

Coefficients:
                           Estimate Std. Error
(Intercept)                 -0.1807     0.5996
homozygote(c104t, "C")TRUE   1.0203     0.2290
allele.count(a1691g, "G")   -0.0905     0.1175
c2249tT/C                    0.4291     0.6873
c2249tT/T                    0.3476     0.5848
                           t value Pr(>|t|)
(Intercept)                  -0.30     0.76
```

```
homozygote(c104t, "C")TRUE    4.46  2.3e-05 ***
allele.count(a1691g, "G")    -0.77     0.44
c2249tT/C                     0.62     0.53
c2249tT/T                     0.59     0.55
---
Signif. codes:  0 '***' 0.001 '**' 0.01
                '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.1 on 95 degrees of
                         freedom
Multiple R-Squared: 0.176,
Adjusted R-squared: 0.141
F-statistic: 5.06 on 4 and 95 DF,
             p-value: 0.000969
```

Conclusion

The current release of the **genetics** package, 1.0.0, provides a complete set of classes and methods for handling single-locus genetic data as well as functions for computing and testing for departure from Hardy-Weinberg and linkage disequilibrium using a variety of estimators.

As noted earlier, Friedrich Leisch and I collaborated on the design of the data structures. While I was primarily motivated by the desire to provide a natural way to include single-locus genetic variables in statistical models, Fritz also wanted to support multiple genetic changes spread across one or more genes. As of the current version, my goal has largely been realized, but more work is necessary to fully support Fritz's goal.

In the future I intend to add functions to perform haplotype imputation and generate standard genetics plots.

I would like to thank Friedrich Leisch for his assistance in designing the genotype data structure, David Duffy for contributing the code for the `gregarious` and HWE.exact functions, and Michael Man for error reports and helpful discussion.

I welcome comments and contributions.

Gregory R. Warnes
Pfizer Global Research and Development
gregory_r_warnes@groton.pfizer.com

Variance Inflation Factors

by Jürgen Groß

A short discussion of centered and uncentered variance inflation factors. It is recommended to report both types of variance inflation for a linear model.

Centered variance inflation factors

A popular method in the classical linear regression model

$$\boldsymbol{y} = \underset{n\times p}{\boldsymbol{X}}\boldsymbol{\beta} + \boldsymbol{\varepsilon},\ \boldsymbol{X} = (\boldsymbol{1}_n, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_p),\ \boldsymbol{\varepsilon} \sim (\boldsymbol{0}, \sigma^2 \boldsymbol{I}_n),$$

to diagnose collinearity is to measure how much the variance $\text{var}(\widehat{\boldsymbol{\beta}}_j)$ of the j-th element $\widehat{\boldsymbol{\beta}}_j$ of the ordinary least squares estimator $\widehat{\boldsymbol{\beta}} = (\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{y}$ is inflated by near linear dependencies of the columns of $\boldsymbol{X}$. This is done by putting $\text{var}(\widehat{\boldsymbol{\beta}}_j)$ in relation to the variance of $\widehat{\boldsymbol{\beta}}_j$ as it would have been when

(a) the nonconstant columns $\boldsymbol{x}_2, \ldots, \boldsymbol{x}_p$ of $\boldsymbol{X}$ had been centered and

(b) the centered columns had been orthogonal to each other.

This variance is given by

$$v_j = \frac{\sigma^2}{\boldsymbol{x}_j'\boldsymbol{C}\boldsymbol{x}_j}, \quad \boldsymbol{C} = \boldsymbol{I}_n - \frac{1}{n}\boldsymbol{1}_n\boldsymbol{1}_n',$$

where $\boldsymbol{C}$ is the usual centering matrix. Then the *centered variance inflation factor* for the j-th variable is

$$\text{VIF}_j = \frac{\text{var}(\widehat{\boldsymbol{\beta}}_j)}{v_j} = \sigma^{-2}\,\text{var}(\widehat{\boldsymbol{\beta}}_j)\boldsymbol{x}_j'\boldsymbol{C}\boldsymbol{x}_j, \quad j = 2, \ldots, p.$$

It can also be written in the form

$$\text{VIF}_j = \frac{1}{1 - \widetilde{R}_j^2},$$

where

$$\widetilde{R}_j^2 = 1 - \frac{\boldsymbol{x}_j'\boldsymbol{M}_j\boldsymbol{x}_j}{\boldsymbol{x}_j'\boldsymbol{C}\boldsymbol{x}_j}, \quad \boldsymbol{M}_j = \boldsymbol{I}_n - \boldsymbol{X}_j(\boldsymbol{X}_j'\boldsymbol{X}_j)^{-1}\boldsymbol{X}_j',$$

is the centered coefficient of determination when the variable x_j is regressed on the remaining independent variables (including the intercept). The matrix $\boldsymbol{X}_j$ is obtained from $\boldsymbol{X}$ by deleting its j-th column. From this it is also intuitively clear that the centered VIF only works satisfactory in a model *with* intercept.

The centered variance inflation factor (as well as a generalized variance-inflation factor) is implemented in the R package **car** by Fox (2002), see also Fox (1997), and can also be computed with the function listed below, which has been communicated by Venables (2002) via the r-help mailing list.

```
> vif <- function(object, ...)
    UseMethod("vif")
```

```
> vif.default <- function(object, ...)
    stop("No default method for vif. Sorry.")

> vif.lm <- function(object, ...) {
    V <- summary(object)$cov.unscaled
    Vi <- crossprod(model.matrix(object))
    nam <- names(coef(object))
    k <- match("(Intercept)", nam,
      nomatch = FALSE)
    if(k) {
      v1 <- diag(V)[-k]
      v2 <- diag(Vi)[-k] - Vi[k, -k]^2 / Vi[k, k]
      nam <- nam[-k]
    }
    else {
      v1 <- diag(V)
      v2 <- diag(Vi)
      warning(paste("No intercept term",
        "detected. Results may surprise."))
    }
    structure(v1 * v2, names = nam)
  }
```

Uncentered variance inflation factors

As pointed out by Belsley (1991, p. 28/29) the centered VIF can be disadvantageous when collinearity effects are related to the intercept. To illustrate this point let us consider the following small ad-hoc example:

```
> n <- 50
> x2 <- 5 + rnorm(n, 0, 0.1)
> x3 <- 1:n
> x4 <- 10 + x2 + x3 + rnorm(n, 0, 10)
> x5 <- (x3/100)^5 * 100
> y <-  10 + 4 + x2 + 1*x3 + 2*x4 + 4*x5 +
    rnorm(n, 0, 50)
> cl.lm <- lm(y ~ x2 + x3 + x4 + x5)
```

Then centered variance inflation factors obtained via `vif.lm(cl.lm)` are

```
      x2       x3       x4       x5
1.017126 6.774374 4.329862 3.122899
```

Several runs produce similar results. Usually values greater than 10 are said to indicate harmful collinearity, which is not the case above, although the VIFs report moderate collinearity related to x_3 and x_4. Since the smallest possible VIF is 1, the above result seems to indicate that x_2 is completely unharmed by collinearity.

No harmful collinearity in the model when the second column x_2 of the matrix $\boldsymbol{X}$ is nearly five times the first column and the scaled condition number, see Belsley (1991, Sec. 3.3), is 147.4653?

As a matter of fact, the centered VIF requires an intercept in the model but at the same time denies the status of the intercept as an independent 'variable' being possibly related to collinearity effects.

Now as indicated by Belsley (1991), an alternative way to measure variance inflation is simply to apply the classical (uncentered) coefficient of determination

$$R_j^2 = 1 - \frac{\boldsymbol{x}_j' \boldsymbol{M}_j \boldsymbol{x}_j}{\boldsymbol{x}_j' \boldsymbol{x}_j}$$

instead of the centered $\widetilde{R}_j^2$. This can also be done for the intercept as an independent 'variable'. For the above example we obtain

```
(Intercept)       x2       x3       x4       x5
   2540.378 2510.568 27.92701 24.89071 4.518498
```

as uncentered VIFs, revealing that the intercept and x_2 are strongly effected by collinearity, which is in fact the case.

It should be noted that $\boldsymbol{x}_j' \boldsymbol{C} \boldsymbol{x}_j \leq \boldsymbol{x}_j' \boldsymbol{x}_j$, so that always $\widetilde{R}_j^2 \leq R_j^2$. Hence the usual critical value $\widetilde{R}_j^2 > 0.9$ (corresponding to centered VIF > 10) for harmful collinearity should be greater when the uncentered VIFs are regarded.

Conclusion

When we use collinearity analysis for finding a possible relationship between impreciseness of estimation and near linear dependencies, then an intercept in the model should not be excluded from the analysis, since the corresponding parameter is as important as any other parameter (e.g. for prediction purposes). See also (Belsley, 1991, Sec. 6.3, 6.8) for a discussion of mean centering and the constant term in connection with collinearity.

By reporting both, centered and uncentered VIFs for a linear model, we can obtain an impression about the possible involvement of the intercept in collinearity effects. This can easily be done by complementing the above `vif.lm` function, since the uncentered VIFs can be computed as in the `else` part of the function, namely as `diag(V) * diag(Vi)`. By this, the uncentered VIFs are computed anyway and not only in the case that no intercept is in the model.

```
> vif.lm <- function(object, ...) {
    V <- summary(object)$cov.unscaled
    Vi <- crossprod(model.matrix(object))
    nam <- names(coef(object))
    k <- match("(Intercept)", nam,
      nomatch = FALSE)
    v1 <- diag(V)
    v2 <- diag(Vi)
    uc.struct <- structure(v1 * v2, names = nam)
    if(k) {
      v1 <- diag(V)[-k]
      v2 <- diag(Vi)[-k] - Vi[k, -k]^2 / Vi[k, k]
      nam <- nam[-k]
      c.struct <- structure(v1 * v2, names = nam)
      return(Centered.VIF = c.struct,
        Uncentered.VIF = uc.struct)
    }
```

```
    else{
      warning(paste("No intercept term",
        "detected. Uncentered VIFs computed."))
      return(Uncentered.VIF = uc.struct)
    }
  }
```

The user of such a function should allow greater critical values for the uncentered than for the centered VIFs.

Bibliography

D. A. Belsley (1991). *Conditioning Diagnostics. Collinearity and Weak Data in Regression*. Wiley, New York. 14

J. Fox (1997). *Applied Regression, Linear Models, and Related Methods*. Sage Publications, Thousand Oaks. 13

J. Fox (2002). *An R and S-Plus Companion to Applied Regression*. Sage Publications, Thousand Oaks. 13

W. N. Venables (2002). *[R] RE: [S] VIF Variance Inflation Factor*. http://www.R-project.org/nocvs/mail/r-help/2002/8566.html. 13

Jürgen Groß
University of Dortmund
Vogelpothsweg 87
D-44221 Dortmund, Germany
gross@statistik.uni-dortmund.de

Building Microsoft Windows Versions of R and R packages under Intel Linux

A Package Developer's Tool

by Jun Yan and A.J. Rossini

It is simple to build R and R packages for Microsoft Windows from an ix86 Linux machine. This is very useful to package developers who are familiar with Unix tools and feel that widespread dissemination of their work is important. The resulting R binaries and binary library packages require only a minimal amount of work to install under Microsoft Windows. While testing is always important for quality assurance and control, we have found that the procedure usually results in reliable packages. This document provides instructions for obtaining, installing, and using the cross-compilation tools.

These steps have been put into a Makefile, which accompanies this document, for automating the process. The Makefile is available from the contributed documentation area on Comprehensive R Archive Network (CRAN). The current version has been tested with R-1.7.0.

For the impatient and trusting, if a current version of Linux R is in the search path, then

```
make CrossCompileBuild
```

will run all Makefile targets described up to the section, *Building R Packages and Bundles*. This assumes: (1) you have the Makefile in a directory `RCrossBuild` (empty except for the makefile), and (2) that `./RCrossBuild` is your current working directory. After this, one should manually set up the packages of interest for building, though the makefile will still help with many important steps. We describe this in detail in the section on *Building R Packages and Bundles*.

Setting up the build area

We first create a separate directory for R cross-building, say `RCrossBuild` and will put the Makefile into this directory. From now on this directory is stored in the environment variable $RCB. Make sure the file name is `Makefile`, unless you are familiar with using the **make** program with other control file names.

Create work area

The following steps should take place in the `RCrossBuild` directory. Within this directory, the Makefile assumes the following subdirectories either exist or can be created, for use as described:

- `downloads` is the location to store all the sources needed for cross-building.
- `cross-tools` is the location to unpack the cross-building tools.
- `WinR` is the location to unpack the R source and cross-build R.
- `LinuxR` is the location to unpack the R source and build a fresh Linux R, which is only needed if your system doesn't have the current version of R.

- `pkgsrc` is the location of package sources (tarred and gzipped from `R CMD build`) that need to cross-build.
- `WinRlibs` is the location to put the resulting Windows binaries of the packages.

These directories can be changed by modifying the `Makefile`. One may find what exactly each step does by looking into the `Makefile`.

Download tools and sources

```
make down
```

This command downloads the i386-mingw32 cross-compilers and R source (starting from R-1.7.0, other sources that used to be necessary for cross building, `pcre` and `bzip2`, are no longer needed). It places all sources into a separate directory called `RCrossBuild/downloads`. The **wget** program is used to get all needed sources. Other downloading tools such as **curl** or **links/elinks** can be used as well. We place these sources into the `RCrossBuild/downloads` directory. The URLs of these sources are available in file `src/gnuwin32/INSTALL` which is located within the R source tree (or, see the `Makefile` associated with this document).

Cross-tools setup

```
make xtools
```

This rule creates a separate subdirectory `cross-tools` in the build area for the cross-tools. It unpacks the cross-compiler tools into the `cross-tools` subdirectory.

Prepare source code

```
make prepsrc
```

This rule creates a separate subdirectory `WinR` to carry out the cross-building process of R and unpacks the R sources into it. As of 1.7.0, the source for `pcre` and `bzip2` are included in the R source tar archive; before that, we needed to worry about them.

Build Linux R if needed

```
make LinuxR
```

This step is required, as of R-1.7.0, if you don't have a current version of Linux R on your system and you don't have the permission to install one for system wide use. This rule will build and install a Linux R in `$RCB/LinuxR/R`.

Building R

Configuring

If a current Linux R is available from the system and on your search path, then run the following command:

```
make mkrules
```

This rule modifies the file `src/gnuwin32/Mkrules` from the R source tree to set `BUILD=CROSS` and `HEADER=$RCB/cross-tools/mingw32/include`.

If a current Linux R is not available from the system, and a Linux R has just been built by `make LinuxR` from the end of the last section:

```
make LinuxFresh=YES mkrules
```

This rule will set `R_EXE=$RCB/LinuxR/R/bin/R`, in addition to the variables above.

Compiling

Now we can cross-compile R:

```
make R
```

The environment variable $PATH is modified in this make target to ensure that the cross-tools are at the beginning of the search path. This step as well as initiation of the compile process is accomplished by the `R` makefile target. This may take a while to finish. If everything goes smoothly, a compressed file `Win-R-1.7.0.tgz` will be created in the `RCrossBuild/WinR` directory.

This gzip'd tar-file contains the executables and supporting files for R which will run on a Microsoft Windows machine. To install, transfer and unpack it on the desired machine. Since there is no InstallShield-style installer executable, one will have to manually create any desired desktop shortcuts to the executables in the `bin` directory, such as `Rgui.exe`. Remember, though, this isn't necessarily the same as the R for Windows version on CRAN!

Building R packages and bundles

Now we have reached the point of interest. As one might recall, the primary goal of this document is to be able to build binary packages for R libraries which can be simply installed for R running on Microsoft Windows. All the work up to this point simply obtains the required working build of R for Microsoft Windows!

Now, create the `pkgsrc` subdirectory to put the package sources into and the `WinRlibs` subdirectory to put the windows binaries into.

We will use the `geepack` package as an example for cross-building. First, put the source `geepack_0.2-4.tar.gz` into the subdirectory `pkgsrc`, and then do

```
make pkg-geepack_0.2-4
```

If there is no error, the Windows binary `geepack.zip` will be created in the `WinRlibs` subdirectory, which is then ready to be shipped to a Windows machine for installation.

We can easily build bundled packages as well. For example, to build the packages in bundle `VR`, we place the source `VR_7.0-11.tar.gz` into the `pkgsrc` subdirectory, and then do

```
make bundle-VR_7.0-11
```

The Windows binaries of packages `MASS`, `class`, `nnet`, and `spatial` in the `VR` bundle will appear in the `WinRlibs` subdirectory.

This Makefile assumes a tarred and gzipped source for an R package, which ends with "`.tar.gz`". This is usually created through the `R CMD build` command. It takes the version number together with the package name.

The Makefile

The associated Makefile is used to automate many of the steps. Since many Linux distributions come with the **make** utility as part of their installation, it hopefully will help rather than confuse people cross-compiling R. The Makefile is written in a format similar to shell commands in order to show what exactly each step does.

The commands can also be cut-and-pasted out of the Makefile with minor modification (such as, change $$ to $ for environmental variable names), and run manually.

Possible pitfalls

We have very little experience with cross-building packages (for instance, Matrix) that depend on external libraries such as atlas, blas, lapack, or Java libraries. Native Windows building, or at least a substantial amount of testing, may be required in these cases. It is worth experimenting, though!

Acknowledgments

We are grateful to Ben Bolker, Stephen Eglen, and Brian Ripley for helpful discussions.

Jun Yan
University of Wisconsin–Madison, U.S.A.
jyan@stat.wisc.edu

A.J. Rossini
University of Washington, U.S.A.
rossini@u.washington.edu

Analysing Survey Data in R

by Thomas Lumley

Introduction

Survey statistics has always been a somewhat specialised area due in part to its view of the world. In the rest of the statistical world data are random and we model their distributions. In traditional survey statistics the population data are fixed and only the sampling is random. The advantage of this 'design-based' approach is that the sampling, unlike the population, is under the researcher's control.

The basic question of survey statistics is

> If we did exactly this analysis on the whole population, what result would we get?

If individuals were sampled independently with equal probability the answer would be very straightforward. They aren't, and it isn't.

To simplify the operations of a large survey it is routine to sample in clusters. For example, a random sample of towns or counties can be chosen and then individuals or families sampled from within those areas. There can be multiple levels of cluster sampling, eg, individuals within families within towns. Cluster sampling often results in people having an unequal chance of being included, for example, the same number of individuals might be surveyed regardless of the size of the town.

For statistical efficiency and to make the results more convincing it is also usual to stratify the population and sample a prespecified number of individuals or clusters from each stratum, ensuring that all strata are fairly represented. Strata for a national survey might divide the country into geographical regions and then subdivide into rural and urban. Ensuring that smaller strata are well represented may also involve sampling with unequal probabilities.

Finally, unequal probabilities might be deliberately used to increase the representation of groups that are small or particularly important. In US health surveys there is often interest in the health of the poor and of racial and ethnic minority groups, who might thus be oversampled.

The resulting sample may be very different in structure from the population, but it is different in ways that are precisely known and can be accounted for in the analysis.

Weighted estimation

The major technique is inverse-probability weighting to estimate population totals. If the probability of sampling individual i is π_i and the value of some variable or statistic for that person is Y_i then an unbiased estimate of the population total is

$$\sum_i \frac{1}{\pi_i} Y_i$$

regardless of clustering or stratification. This is called the Horvitz–Thompson estimator (Horvitz & Thompson, 1951). Inverse-probability weighting can be used in an obvious way to compute population means and higher moments. Less obviously, it can be used to estimate the population version of loglikelihoods and estimating functions, thus allowing almost any standard models to be fitted. The resulting estimates answer the basic survey question: they are consistent estimators of the result that would be obtained if the same analysis was done to the whole population.

It is important to note that estimates obtained by maximising an inverse-probability weighted loglikelihood are not in general maximum likelihood estimates under the model whose loglikelihood is being used. In some cases semiparametric maximum likelihood estimators are known that are substantially more efficient than the survey-weighted estimators if the model is correctly specified. An important example is a two-phase study where a large sample is taken at the first stage and then further variables are measured on a subsample.

Standard errors

The standard errors that result from the more common variance-weighted estimation are incorrect for probability weighting and in any case are model-based and so do not answer the basic question. To make matters worse, stratified and clustered sampling also affect the variance, the former decreasing it and the latter (typically) increasing it relative to independent sampling.

The formulas for standard error estimation are developed by first considering the variance of a sum or mean under independent unequal probability sampling. This can then be extended to cluster sampling, where the clusters are independent, and to stratified sampling by adding variance estimates from each stratum. The formulas are similar to the 'sandwich variances' used in longitudinal and clustered data (but not quite the same).

If we index strata by s, clusters by c and observations by i then

$$\hat{\mathrm{var}}[S] = \sum_s \frac{n_s}{n_s - 1} \sum_{c,i} \left(\frac{1}{\pi_{sci}} \left(Y_{sci} - \bar{Y}_s \right) \right)^2$$

where S is the weighted sum, $\bar{Y}_s$ are weighted stratum means, and n_s is the number of clusters in stratum s.

Standard errors for statistics other than means are developed from a first-order Taylor series expansion, that is, they use the same formula applied to the mean of the estimating functions. The computations for variances of sums are performed in the `svyCprod` function, ensuring that they are consistent across the survey functions.

Subsetting surveys

Estimates on a subset of a survey require some care. Firstly, it is important to ensure that the correct weight, cluster and stratum information is kept matched to each observation. Secondly, it is not correct simply to analyse a subset as if it were a designed survey of its own.

The correct analysis corresponds approximately to setting the weight to zero for observations not in the subset. This is not how it is implemented, since observations with zero weight still contribute to constraints in R functions such as `glm`, and they still take up memory. Instead, the `survey.design` object keeps track of how many clusters (PSUs) were originally present in each stratum and `svyCprod` uses this to adjust the variance.

Example

The examples in this article use data from the National Health Interview Study (NHIS 2001) conducted by the US National Center for Health Statistics. The data and documentation are available from `http://www.cdc.gov/nchs/nhis.htm`. I used NCHS-supplied SPSS files to read the data and then `read.spss` in the `foreign` package to load them into R. Unfortunately the dataset is too big to include these examples in the `survey` package — they would increase the size of the package by a couple of orders of magnitude.

The survey package

In order to keep the stratum, cluster, and probability information associated with the correct observations the `survey` package uses a `survey.design` object created by the `svydesign` function. A simple example call is

```
imdsgn<-svydesign(id=~PSU,strata=~STRATUM,
            weights=~WTFA.IM,
            data=immunize,
            nest=TRUE)
```

The data for this, the immunization section of NHIS, are in the data frame `immunize`. The strata are specified by the `STRATUM` variable, and the inverse probability weights are in the `WTFA.IM` variable. The statement specifies only one level of clustering, by `PSU`. The `nest=TRUE` option asserts that clusters are nested in strata so that two clusters with the same `psuid` in different strata are actually different clusters.

In fact there are further levels of clustering and it would be possible to write

```
imdsgn<-svydesign(id=~PSU+HHX+FMX+PX,
            strata=~STRATUM,
            weights=~WTFA.IM,
            data=immunize,
            nest=TRUE)
```

to indicate the sampling of households, families, and individuals. This would not affect the analysis, which depends only on the largest level of clustering. The main reason to provide this option is to allow sampling probabilities for each level to be supplied instead of weights.

The `strata` argument can have multiple terms: eg `strata= region+rural` specifying strata defined by the interaction of `region` and `rural`, but NCHS studies provide a single stratum variable so this is not needed. Finally, a finite population correction to variances can be specified with the `fpc` argument, which gives either the total population size or the overall sampling fraction of top-level clusters (PSUs) in each stratum.

Note that variables to be looked up in the supplied data frame are all specified by formulas, removing the scoping ambiguities in some modelling functions.

The resulting `survey.design` object has methods for subscripting and `na.action` as well as the usual print and summary. The print method gives some descriptions of the design, such as the number of largest level clusters (PSUs) in each stratum, the distribution of sampling probabilities and the names of all the data variables. In this immunisation study the sampling probabilities vary by a factor of 100, so ignoring the weights may be very misleading.

Analyses

Suppose we want to see how many children have had their recommended polio immmunisations recorded. The command

```
svytable(~AGE.P+POLCT.C, design=imdsgn)
```

produces a table of number of polio immunisations by age in years for children with good immunisation data, scaled by the sampling weights so that it corresponds to the US population. To fit the table more easily on the page, and since three doses is regarded as sufficient we can do

```
> svytable(~AGE.P+pmin(3,POLCT.C),design=imdsgn)
     pmin(3, POLCT.C)
AGE.P      0      1      2       3
    0 473079 411177 655211  276013
    1 162985  72498 365445  863515
    2 144000  84519 126126 1057654
    3  89108  68925 110523 1123405
    4 133902 111098 128061 1069026
    5 137122  31668  19027 1121521
    6 127487  38406  51318  838825
```

where the last column refers to 3 or more doses. For ages 3 and older, just over 80% of children have received 3 or more doses and roughly 10% have received none.

To examine whether this varies by city size we could tabulate

```
svytable(~AGE.P+pmin(3,POLCT.C)+MSASIZEP,
          design=imdsgn)
```

where `MSASIZEP` is the size of the 'metropolitan statistical area' in which the child lives, with categories ranging from 5,000,000+ to under 250,000, and a final category for children who don't live in a metropolitan statistical area. As `MSASIZEP` has 7 categories the data end up fairly sparse. A regression model might be more useful.

Regression models

The `svyglm` and `svycoxph` functions have similar syntax to their non-survey counterparts, but use a `design` rather than a `data` argument. For simplicity of implementation they do require that all variables in the model be found in the `design` object, rather than floating free in the calling environment.

To look at associations with city size we fit the logistic regression models in Figure 1. The resulting `svyglm` objects have a summary method similar to that for `glm`. There isn't a consistent association, but category 2: cities from 2.5 million to 5 million people, does show some evidence of lower vaccination rates. Similar analyses using family income don't show any real evidence of an assocation.

General weighted likelihood estimation

The `svymle` function maximises a specified weighted likelihood. It takes as arguments a loglikelihood function and formulas or fixed values for each parameter in the likelihood. For example, a linear regression could be implemented by the code in Figure 2. In this example the `log=TRUE` argument is passed to `dnorm` and is the only fixed argument. The other two arguments, `mean` and `sd`, are specified by

```
svyglm(I(POLCT.C>=3)~factor(AGE.P)+factor(MSASIZEP), design=imdsgn, family=binomial)
svyglm(I(POLCT.C>=3)~factor(AGE.P)+MSASIZEP,  design=imdsgn, family=binomial)
```

Figure 1: Logistic regression models for vaccination status

```
gr <- function(x,mean,sd,log)
     dm <- 2*(x - mean)/(2*sd^2)
     ds <- (x-mean)^2*(2*(2 * sd))/(2*sd^2)^2 - sqrt(2*pi)/(sd*sqrt(2*pi))
     cbind(dm,ds)

m2 <- svymle(loglike=dnorm,gradient=gr, design=dxi,
             formulas=list(mean=y~x+z, sd=~1),
             start=list(c(2,5,0),  c(4)),
             log=TRUE)
```

Figure 2: Weighted maximum likelihood estimation

formulas. Exactly one of these formulas should have a left-hand side specifying the response variable.

It is necessary to specify the gradient in order to get survey-weighted standard errors. If the gradient is not specified you can get standard errors based on the information matrix. With independent weighted sampling the information matrix standard errors will be accurate if the model is correctly specified. As is always the case with general-purpose optimisation it is important to have reasonable starting values; you could often get them from an analysis that ignored the survey design.

Extending the survey package

It is easy to extend the survey package to include any new class of model where weighted estimation is possible. The `svycoxph` function shows how this is done.

1. Create a call to the underlying model (`coxph`), adding a `weights` argument with weights $1/\pi_i$ (or a rescaled version).
2. Evaluate the call
3. Extract the one-step delta-betas Δ_i or compute them from the information matrix I and score contributions U_i as $\Delta = I^{-1}U_i$
4. Pass Δ_i and the cluster, strata and finite population correction information to `svyCprod` to compute the variance.
5. Add a `"svywhatever"` class to the object and a copy of the design object
6. Override the `print` and `print.summary` methods to include `print(x$design, varnames=FALSE, design.summaries=FALSE)`
7. Override any likelihood extraction methods to fail

It may also be necessary to override other methods such as `residuals`, as is shown for Pearson residuals in `residuals.svyglm`.

A second important type of extension is to add prediction methods for small-area extrapolation of surveys. That is, given some variables measured in a small geographic area (a new cluster), predict other variables using the regression relationship from the whole survey. The `predict` methods for survey regression methods current give predictions only for individuals.

Extensions in third direction would handle different sorts of survey design. The current software cannot handle surveys where strata are defined within clusters, partly because I don't know what the right variance formula is.

Summary

Analysis of complex sample surveys used to require expensive software on expensive mainframe hardware. Computers have advanced sufficiently for a reasonably powerful PC to analyze data from large national surveys such as NHIS, and R makes it possible to write a useful survey analysis package in a fairly short time. Feedback from practising survey statisticians on this package would be particularly welcome.

References

Horvitz DG, Thompson DJ (1951). A Generalization of Sampling without Replacement from a Finite Universe. *Journal of the American Statistical Association*. 47, 663-685.

Thomas Lumley
Biostatistics, University of Washington
tlumley@u.washington.edu

Computational Gains Using RPVM on a Beowulf Cluster

by Brett Carson, Robert Murison and Ian A. Mason.

Introduction

The Beowulf cluster Becker et al. (1995); Scyld Computing Corporation (1998) is a recent advance in computing technology that harnesses the power of a network of desktop computers using communication software such as PVM Geist et al. (1994) and MPI Message Passing Interface Forum (1997). Whilst the potential of a computing cluster is obvious, expertise in programming is still developing in the statistical community.

Recent articles in R-news Li and Rossini (2001) and Yu (2002) entice statistical programmers to consider whether their solutions could be effectively calculated in parallel. Another R package, SNOW Tierney (2002); Rossini et al. (2003) aims to skillfully provide a wrapper interface to these packages, independent of the underlying cluster communication method used in parallel computing. This article concentrates on RPVM and wishes to build upon the contribution of Li and Rossini (2001) by taking an example with obvious orthogonal components and detailing the R code necessary to allocate the computations to each node of the cluster. The statistical technique used to motivate our RPVM application is the gene-shaving algorithm Hastie et al. (2000b,a) for which S-PLUS code has been written by Do and Wen (2002) to perform the calculations serially.

The first section is a brief description of the Beowulf cluster used to run the R programs discussed in this paper. This is followed by an explanation of the gene-shaving algorithm, identifying the opportunities for parallel computing of bootstrap estimates of the "strength" of a cluster and the rendering of each matrix row orthogonal to the "eigen-gene". The code for spawning child processes is then explained comprehensively and the conclusion compares the speed of RPVM on the Beowulf to serial computing.

A parallel computing environment

The Beowulf cluster Becker et al. (1995); Scyld Computing Corporation (1998) used to perform the parallel computation outlined in the following sections is a sixteen node homogeneous cluster. Each node consists of an AMD XP1800+ with 1.5Gb of memory, running Redhat Linux 7.3. The nodes are connected by 100Mbps Ethernet, and are accessible via the dual processor AMD XP1800+ front-end. Each node has installed versions of R, PVM Geist et al. (1994) and R's PVM wrapper, RPVM Li and Rossini (2001).

Gene-shaving

The gene-shaving algorithm Hastie et al. (2000b,a) is one of the many DNA micro-array gene clustering techniques to be developed in recent times. The aim is to find clusters with small variance between genes, and large variance between samples Hastie et al. (2000b). The algorithm consists of a number of steps:

1. The process starts with the matrix $X_{N,p}$ which represents expressions of genes in the micro array. Each row of X is data from a gene, the columns contain samples. N is usually of size 10^3 and p usually no greater than 10^2.

2. The leading principal component of X is calculated and termed the "eigen-gene".

3. The correlation of each row with the eigen-gene is calculated and the rows yielding the lowest 10% of R^2 are shaved. This process is repeated to produce successive reduced matrices $X_1^*, \ldots X_N^*$. There are $[0.9N]$ genes in X_1^*, $[0.81N]$ in X_2^* and 1 in X_N^*.

4. The optimal cluster size is selected from $\mathbf{X}^*$. The selection process requires that we estimate the distribution of R^2 by bootstrap and calculate $G_k = R_k^2 - R*_k^2$ where $R*_k^2$ is the mean R^2 from B bootstrap samples of X_k. The statistic G_k is termed the gap statistic and the optimal cluster is that which has the largest gap statistic.

5. Once a cluster has been determined, the process repeats searching for more clusters, after removing the information of previously determined clusters. This is done by "sweeping" the mean gene of the optimal cluster, $\bar{x}_{S_{\hat{k}}}$, from each row of X,

$$newX_{[i,]} = X_{[i,]} \underbrace{(I - \bar{x}_{S_{\hat{k}}}(\bar{x}_{S_{\hat{k}}}^T \bar{x}_{S_{\hat{k}}})^{-1} \bar{x}_{S_{\hat{k}}}^T)}_{IPx}$$

6. The next optimal cluster is found by repeating the process using $newX$.

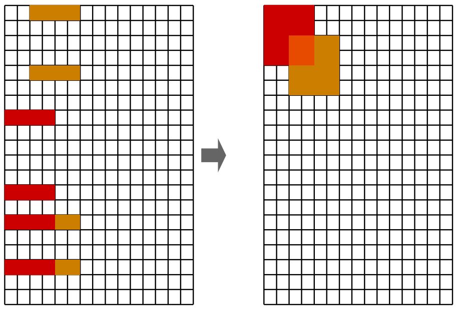


Figure 1: Clustering of genes

The bootstrap and orthogonalization processes are candidates for parallel computing. Figure 1 illustrates the overall gene-shaving process by showing the discovery of clusters of genes.

Parallel implementation of gene-shaving

There are two steps in the shaving process that can be computed in parallel. Firstly, the bootstrapping of the matrix X, and secondly, the orthogonalization of the matrix X with respect to $\bar{x}_{S_k}$. In general, for an algorithm that can be fully parallelized, we could expect a speedup of no more than 16 on a cluster of 16 nodes. However, given that this particular algorithm will not be fully parallelized, we would not expect such a reduction in computation time. In fact, the cluster will not be fully utilised during the overall computation, as most of the nodes will see significant idle time. Amdahl's Law Amdahl (1967) can be used to calculate the estimated speedup of performing gene-shaving in parallel. Amdahl's Law can be represented in mathematical form by:

$$Speedup = \frac{1}{r_s + \frac{r_p}{n}}$$

where r_s is the proportion of the program computed in serial, r_p is the parallel proportion and n is the number of processors used. In a perfect world, if a particular step computed in parallel were spread over five nodes, the speedup would equal five. This is very rarely the case however, due to speed limiting factors like network load, PVM overheads, communication overheads, CPU load and the like.

The parallel setup

Both the parallel versions of bootstrapping and orthogonalization consist of a main R process (the master), and a number of child processes that the master spawns (the slaves). The master process controls the entire flow of the shaving process. It begins by starting the child processes, which then proceed to wait for the master to send data for processing, and return an answer to the master. There are two varieties of children/slaves:

- Bootstrapping children - each slave task performs one permutation of the bootstrapping process.
- Orthogonalization children - each slave computes a portion of the orthogonalization process.

Figure 2 shows a basic master-slave setup, without any communication between the slaves themselves. The slaves receive some data from the parent, process it, and send the result back to the parent. A more elaborate setup would see the slaves communicating with each other, this is not necessary in this case, as each of the slaves is independent of each other.

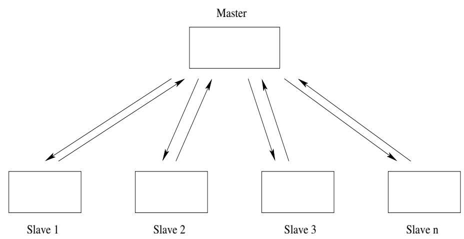


Figure 2: The master-slave paradigm without slave-slave communication.

Bootstrapping

Bootstrapping is used to find $R*^2$ used in the gap statistic outlined in step 4 of the gene-shaving algorithm, to determine whether selected clusters could have arisen by chance. Elements of each row of the matrix are rearranged, which requires N iterations, quite a CPU intensive operation, even more so for larger expression matrices. In the serial R version, this would be done using R's `apply` function, whereby each row of X has the `sample` function applied to it.

The matrix X is transformed into X^* B times by randomly permuting within rows. For each of the B bootstrap samples (indexed by b), the statistic R_b^2 is derived and the mean of $R_b^2, b = 1, B$ is calculated to represent that R^2 which arises by chance only. In RPVM, for each bootstrap sample a task is spawned; for B bootstrap samples we spawn B tasks.

Orthogonalization

The master R process distributes a portion of the matrix X to its children. Each portion will be approximately equal in size, and the number of portions will be equal to the number of slaves spawned for orthogonalization. The master also calculates the value of IPx, as defined in step 5 of the gene-shaving algorithm.

The value of IPx is passed to each child. Each child then applies IPx to each row of X that the master has given it.

There are two ways in which X can be distributed. The matrix can be split into approximately even portions, where the number of portions will be equal to the number of slaves. Alternatively, the matrix can be distributed on a row-by-row basis, where each slave processes a row, sends it back, is given a new row to process and so on. The latter method becomes slow and inefficient, as more sending, receiving, and unpacking of buffers is required. Such a method may become useful where the number of rows of the matrix is small and the number of columns is very large.

The RPVM code

Li and Rossini (2001) have explained the initial steps of starting the PVM daemon and spawning RPVM tasks from within an R program, These steps precede the following code. The full source code is available from http://mcs.une.edu.au/~bcarson/RPVM/.

Bootstrapping

Bootstrapping can be performed in parallel by treating each bootstrap sample as a child process. The number of children is the number of bootstrap samples, (B) and the matrix is copied to each child, $(1, ..., B)$. Child process i permutes elements within each row of X, calculates $R*^2$ and returns this to the master process. The master averages $R*^2$, $i = 1, B$ and computes the gap statistic.

The usual procedure for sending objects to PVM child processes is to initialize the send buffer, pack the buffer with data and send the buffer. Our function par.bootstrap is a function used by the master process and includes .PVM.initsend() to initialize the send buffer, .PVM.pkdblmat(X) to pack the buffer, and .PVM.mcast(boot.children,0) to send the data to all children.

```
par.bootstrap <- function(X){
  brsq <- 0

  tasks <- length(boot.children)

  .PVM.initsend()
  #send a copy of X to everyone
  .PVM.pkdblmat(X)
  .PVM.mcast(boot.children,0)

  #get the R squared values back and
  #take the mean
  for(i in 1:tasks){
    .PVM.recv(boot.children[i],0)
    rsq <- .PVM.upkdouble(1,1)
    brsq <- brsq + (rsq/tasks)
  }
  return(brsq)
}
```

After the master has sent the matrix to the children, it waits for them to return their calculated R^2 values. This is controlled by a simple for loop, as the master receives the answer via .PVM.recv(boot.children[i],0), it is unpacked from the receive buffer with:

```
rsq <- .PVM.upkdouble(1,1)
```

The value obtained is used in calculating an overall mean for all R^2 values returned by the children. In the slave tasks, the matrix is unpacked and a bootstrap permutation is performed, an R^2 value calculated and returned to the master. The following code performs these operations of the children tasks:

```
#receive the matrix
.PVM.recv(.PVM.parent(),0)
X <- .PVM.upkdblmat()

Xb <- X
dd <- dim(X)

#perform the permutation
for(i in 1:dd[1]){
  permute <- sample(1:dd[2],replace=F)
  Xb[i,] <- X[i,permute]
}

#send it back to the master
.PVM.initsend()
.PVM.pkdouble(Dfunct(Xb))
.PVM.send(.PVM.parent(),0)
```

.PVM.upkdblmat() is used to unpack the matrix. After the bootstrap is performed, the send is initialized with .PVM.initsend() (as seen in the master), the result of Dfunct(Xb) (R^2 calculation) is packed into the send buffer, and then the answer is sent with .PVM.send() to the parent of the slave, which will be the master process.

Orthogonalization

The orthogonalization in the master process is handled by the function par.orth. This function takes as arguments the IPx matrix, and the expression matrix X. The first step of this procedure is to broadcast the IPx matrix to all slaves by firstly packing it and then sending it with .PVM.mcast(orth.children,0). The next step is to split the expression matrix into portions and distribute to the slaves. For simplicity, this example divides the matrix by the number of slaves, so row size must be divisible by the number of children. Using even blocks on a homogeneous cluster will work reasonably well, as the load-balancing issues outlined in Rossini et al. (2003) are more relevant to heterogeneous clusters. The final step performed by the master is to receive these portions back from the slaves and reconstruct the matrix. Order of rows is maintained by sending blocks of rows and receiving them back in the order the children sit in the array of task ids (`orth.children`). The following code is the `par.orth` function used by the master process:

```
par.orth <- function(IPx, X){
  rows <- dim(X)[1]

  tasks <- length(orth.children)
  #intialize the send buffer
  .PVM.initsend()
  #pack the IPx matrix and send it to
  #all children
  .PVM.pkdblmat(IPx)
  .PVM.mcast(orth.children,0)

  #divide the X matrix into blocks and
  #send a block to each child
  n.rows = as.integer(rows/tasks)
  for(i in 1:tasks){
    start <- (n.rows * (i-1))+1
    end <- n.rows * i
    if(end > rows)
      end <- rows
    X.part <- X[start:end,]
    .PVM.pkdblmat(X.part)
    .PVM.send(orth.children[i],start)
  }

  #wait for the blocks to be returned
  #and re-construct the matrix
  for(i in 1:tasks){
    .PVM.recv(orth.children[i],0)
    rZ <- .PVM.upkdblmat()
    if(i==1){
      Z <- rZ
    }
    else{
      Z <- rbind(Z,rZ)
    }
  }
  dimnames(Z) <- dimnames(X)
  Z
}
```

The slaves execute the following code, which is quite similar to the code of the bootstrap slaves. Each slave receives both matrices, then performs the orthogonalization and sends back the result to the master:

```
  #wait for the IPx and X matrices to arrive
  .PVM.recv(.PVM.parent(),0)
  IPx <- .PVM.upkdblmat()
  .PVM.recv(.PVM.parent(),-1)
  X.part <- .PVM.upkdblmat()

  Z <- X.part
  #orthogonalize X
  for(i in 1:dim(X.part)[1]){
    Z[i,] <- X.part[i,] %*% IPx
  }
  #send back the new matrix Z
  .PVM.initsend()
  .PVM.pkdblmat(Z)
  .PVM.send(.PVM.parent(),0)
```

Gene-shaving results

Serial gene-shaving

The results for the serial implementation of gene-shaving are a useful guide to the speedup that can be achieved from parallelizing. The proportion of the total computation time the bootstrapping and orthogonalization use is of interest. If these are of significant proportion, they are worth performing in parallel.

The results presented were obtained from running the serial version on a single cluster node. RPVM is not used in the serial version, and the program contains no parallel code at all. The data used are randomly generated matrices of size $N = 1600, 3200, 6400$, and 12800 respectively, with a constant p of size 80, to simulate variable-sized micro-arrays. In each case, the first two clusters are found, and ten permutations used in each bootstrap. Figure 3 is a plot of the overall computation time, the total time of all invocations of the bootstrapping process, and the total time of all invocations of the orthogonalization process.

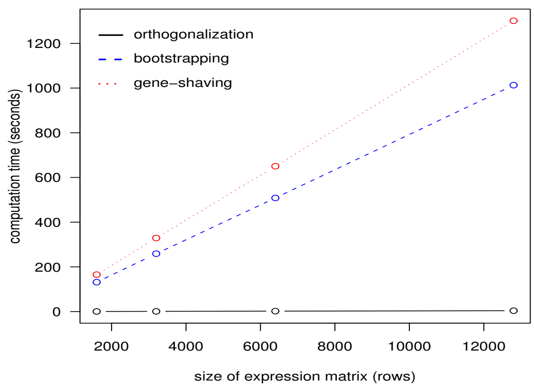


Figure 3: Serial Gene-shaving results

Clearly, the bootstrapping (blue) step is quite time consuming, with the mean proportion being 0.78 of the time taken to perform gene-shaving (red). The orthogonalization (black) equates to no more than 1%, so any speedup through parallelization is not going to have any significant effect on the overall time.

We have two sets of parallel results, one using two cluster nodes, the other using all sixteen nodes of the cluster. In the two node runs we could expect a reduction in the bootstrapping time to be no more than half, i.e a *speedup* of 2. In full cluster runs we could expect a *speedup* of no more than 10, as we can use ten of the sixteen processors (one processor for each permutation). It is now possible to estimate the expected maximum speedup of parallelization with Amdahl's law, as described earlier for expression matrices of size $N = 12800$ and $p = 80$, given the parallel proportion of 0.78.

Application of Amdahl's law fot the cases $(n = 2, 10)$ gives speedups of 1.63 and 3.35 respectively. We have implemented the orthogonalization in parallel, more for demonstration purposes, as it could become useful with very large matrices.

Parallel gene-shaving

The parallel version of gene-shaving only contains parallel code for the bootstrapping and orthogonalization procedures, the rest of the process remains the same as the serial version. Both the runs using two nodes and all sixteen nodes use this same `R` program. In both cases ten tasks are spawned for bootstrapping and sixteen tasks are spawned for orthogonalization. Clearly this is an under-utilisation of the Beowulf cluster in the sixteen node runs, as most of the nodes are going to see quite a lot of idle time, and in fact six of the nodes will not be used to compute the bootstrap, the most `CPU` intensive task of the entire algorithm.

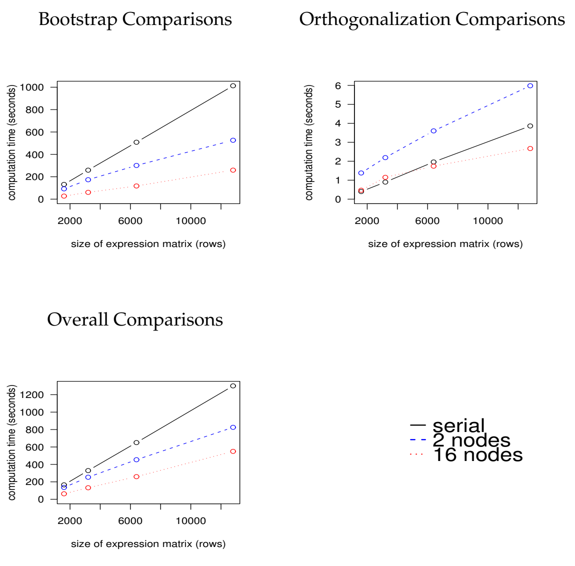


Figure 4: Comparison of computation times for variable sized expression matrices for overall, orthogonalization and bootstrapping under the different environments.

The observed speedup for the two runs of parallel gene-shaving using the $N = 12800$ expression matrix is as follows:

$$S = \frac{T_{serial}}{T_{parallel}}$$

$$S_{2nodes} = \frac{1301.18}{825.34} = 1.6$$

$$S_{16nodes} = \frac{1301.18}{549.59} = 2.4$$

The observed speedup for two nodes is very close to the estimate of 1.63. The results for sixteen nodes are not as impressive, due to greater time spent computing than expected for bootstrapping. Although there is a gain with the orthogonalization here, it is barely noticeable in terms of the whole algorithm. The orthogonalization is only performed once for each cluster found, so it may well become an issue when searching for many clusters on much larger datasets than covered here.

Conclusions

Whilst the gene-shaving algorithm is not completely parallelizable, a large portion of it (almost 80%) of it can be implemented to make use of a Beowulf cluster. This particular problem does not fully utilise the cluster, each node other than the node which runs the master process will see a significant amount of idle time. Regardless, the results presented in this paper show the value and potential of Beowulf clusters in processing large sets of statistical data. A fully parallelizable algorithm which can fully utilise each cluster node will show greater performance over a serial implementation than the particular example we have presented.

Bibliography

Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. In *AFIPS spring joint computer conference*, Sunnyvale, California. IBM. 22

Becker, D. J., Sterling, T., Savarese, D., Dorband, J. E., Ranawak, U. A., and Packer, C. V. (1995). BEOWULF: A PARALLEL WORKSTATION FOR SCIENTIFIC COMPUTATION. In *International Conference on Parallel Processing*. 21

Do, K.-A. and Wen, S. (2002). Documentation. Technical report, Department of Biostatistics, MD Anderson Cancer Center. http://odin.mdacc.tmc.edu/ kim/. 21

Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., and Sunderam, V. (1994). *PVM: Parallel Virtual Machine. A user's guide and tutorial for networked parallel computing*. MIT Press. 21

Hastie, T., Tibshirani, R., Eisen, M., Alizadeh, A., Levy, R., Staudt, L., Chan, W., Botstein, D., and Brown, P. (2000a). 'Gene shaving' as a method for identifying distinct sets of genes with similar expression patterns. *Genome Biology*. 21

Hastie, T., Tibshirani, R., Eisen, M., Brown, P., Scherf, U., Weinstein, J., Alizadeh, A., Staudt, L., and Botstein, D. (2000b). Gene shaving: a new class of clustering methods for expression arrays. Technical report, Stanford University. 21

Li, M. N. and Rossini, A. (2001). RPVM: Cluster statistical computing in R. *R News*, 1(3):4–7. 21, 23

Message Passing Interface Forum (1997). MPI-2: Extensions to the Message-Passing Interface. http://www.mpi-forum.org/docs/docs.html. 21

Rossini, A., Tierney, L., and Li, N. (2003). Simple parallel statistical computing in r. Technical Report Working Paper 193, UW Biostatistics Working Paper Series. http://www.bepress.com/uwbiostat/paper193. 21, 23

Scyld Computing Corporation (1998). Beowulf introduction & overview. http://www.beowulf.org/intro.html. 21

Tierney, L. (2002). Simple Network of Workstations for R. Technical report, School of Statistics, University of Minnesota. http://www.stat.uiowa.edu/ luke/R/cluster/cluster.html. 21

Yu, H. (2002). Rmpi: Parallel statistical computing in R. *R News*, 2(2):10–14. 21

Brett Carson, Robert Murison, Ian A. Mason
The University of New England, Australia
bcarson@turing.une.edu.au
rmurison@turing.une.edu.au
iam@turing.une.edu.au

R Help Desk

Getting Help – R's Help Facilities and Manuals

Uwe Ligges

Introduction

This issue of the *R Help Desk* deals with its probably most fundamental topic: Getting help!

Different amounts of knowledge about R and R related experiences require different levels of help. Therefore a description of the available manuals, help functions within R, and mailing lists will be given, as well as an "instruction" how to use these facilities.

A secondary objective of the article is to point out that manuals, help functions, and mailing list archives are commonly much more comprehensive than answers on a mailing list. In other words, there are good reasons to read manuals, use help functions, and search for information in mailing list archives.

Manuals

The manuals described in this Section are shipped with R (in directory '.../doc/manual' of a regular R installation). Nevertheless, when problems occur before R is installed or the available R version is outdated, recent versions of the manuals are also available from CRAN[1]. Let me cite the "R Installation and Administration" manual by the *R Development Core Team* (R Core, for short; 2003b) as a reference for information on installation and updates not only for R itself, but also for contributed packages.

I think it is not possible to use adequately a programming language and environment like R without reading any introductory material . Fundamentals for using R are described in "An Introduction to R" (Venables et al., 2003), and the "R Data Import/Export" manual (R Core, 2003a). Both manuals are the first references for any programming tasks in R.

Frequently asked questions (and corresponding answers) are collected in the "R FAQ" (Hornik, 2003), an important resource not only for beginners. It is a good idea to look into the "R FAQ" when a question arises that cannot be answered by reading the other manuals cited above. Specific FAQs for Macintosh (by Stefano M. Iacus) and Windows (by Brian D. Ripley) are available as well[2].

Users who gained some experiences might want to write their own functions making use of the language in an efficient manner. The manual "R Language Definition" (R Core, 2003c, still labelled as "draft") is a comprehensive reference manual for a couple of important language aspects, e.g.: objects, working with expressions, working with the language (objects), description of the parser, and *debugging*. I recommend this manual to all users who plan to work regularly with R, since it provides really illuminating insights.

For those rather experienced users who plan to create their own packages, writing help pages and optimize their code, "Writing R Extensions" (R Core,

[1] http://CRAN.R-project.org/
[2] http://CRAN.R-project.org/faqs.html

2003d) is the appropriate reference. It includes also descriptions on interfacing to C, C++, or Fortran code, and compiling and linking this code into R.

Occasionally, `library(help=PackageName)` indicates that package specific manuals (or papers corresponding to packages), so-called vignettes, are available in directory '.../library/PackageName/doc'.

Beside the manuals, several books dealing with R are available these days, e.g.: introductory textbooks, books specialized on certain applications such as regression analysis, and books focussed on programming and the language. Here I do not want to advertise any of those books, because the "optimal" choice of a book depends on the tasks a user is going to perform in R. Nevertheless, I would like to mention Venables and Ripley (2000) as a reference for programming and the language, and Venables and Ripley (2002) as a comprehensive book dealing with several statistical applications and how to apply those in R (and S-PLUS).

Help functions

Access to the documentation on the topic *help* (in this case a function name) is provided by `?help`. The "?" is the probably most frequently used way for getting help, whereas the corresponding function `help()` is rather flexible. As the default, help is shown as formatted text. This can be changed (by arguments to `help()`, or for a whole session with `options()`) to, e.g., HTML (`htmlhelp=TRUE`), if available. The HTML format provides convenient features like links to other related help topics. On Windows, I prefer to get help in Compiled HTML format by setting `options(chmhelp=TRUE)` in file '.Rprofile'.

When exact names of functions or other topics are unknown, the user can search for relevant documentation (matching a given character string in, e.g. name, title, or keyword entries) with `help.search()`. This function allows quite flexible searching using either regular expression or fuzzy matching, see `?help.search` for details. If it is required to find an object by its partial name, `apropos()` is the right choice – it also accepts regular expressions.

Documentation and help in HTML format can also be accessed by `help.start()`. This functions starts a Web browser with an index that links to HTML versions of the manuals mentioned above, the FAQs, a list of available packages of the current installation, several other information, and a page containing a search engine and keywords. On the latter page, the search engine can be used to "search for keywords, function and data names and text in help page titles" of installed packages. The list of keywords is useful when the search using other facilities fails: the user gets a complete list of functions and data sets corresponding to a particular keyword.

Of course, it is not possible to search for functionality of (unknown) contributed packages or documentation packages (and functions or data sets defined therein) that are not available in the current installation of R. A listing of contributed packages and their titles is given on CRAN, as well as their function indices and reference manuals. Nevertheless, it might be still hard to find the function or package of interest. For that purpose, an R site search[3] provided by Jonathan Baron is linked on the search page of CRAN. Help files, manuals, and mailing list archives (see below) can be searched.

Mailing lists

Sometimes manuals and help functions do not answer a question, or there is need for a discussion. For these purposes there are the mailing lists[4] *r-announce* for important announcements, *r-help* for questions and answers about problems and solutions using R, and *r-devel* for the development of R.

Obviously, *r-help* is the appropriate list for asking questions that are not covered in the manuals (including the "R FAQ") or the help pages, and regularly those questions are answered within a few hours (or even minutes).

Before asking, it is a good idea to look into the archives of the mailing list.[5] These archives contain a huge amount of information and may be much more helpful than a direct answer on a question, because most questions already have been asked – and answered – several times (focussing on slightly different aspects, hence these are illuminating a bigger context).

It is really easy to ask questions on *r-help*, but I would like to point out the advantage of reading manuals *before* asking: Of course, reading (a section of) some manual takes much more time than asking, and answers from voluntary help providers of *r-help* will probably solve the recent problem of the asker, but quite similar problems will certainly arise pretty soon. Instead, the time one spends reading the manuals will be saved in subsequent programming tasks, because many side aspects (e.g. convenient and powerful functions, programming tricks, etc.) will be learned from which one benefits in other applications.

[3] `http://finzi.psych.upenn.edu/search.html`, see also `http://CRAN.R-project.org/search.html`
[4] accessible via `http://www.R-project.org/mail.html`
[5] archives are searchable, see `http://CRAN.R-project.org/search.html`

Bibliography

Hornik, K. (2003): *The R FAQ*, Version 1.7-3. http://www.ci.tuwien.ac.at/~hornik/R/ ISBN 3-901167-51-X. 26

R Development Core Team (2003a): *R Data Import/Export*. URL http://CRAN.R-project.org/manuals.html. ISBN 3-901167-53-6. 26

R Development Core Team (2003b): *R Installation and Administration*. URL http://CRAN.R-project.org/manuals.html. ISBN 3-901167-52-8. 26

R Development Core Team (2003c): *R Language Definition*. URL http://CRAN.R-project.org/manuals.html. ISBN 3-901167-56-0. 26

R Development Core Team (2003d): *Writing R Extensions*. URL http://CRAN.R-project.org/manuals.html. ISBN 3-901167-54-4. 26

Venables, W. N. and Ripley, B. D. (2000): *S Programming*. New York: Springer-Verlag. 27

Venables, W. N. and Ripley, B. D. (2002): *Modern Applied Statistics with S*. New York: Springer-Verlag, 4th edition. 27

Venables, W. N., Smith, D. M., and the R Development Core Team (2003): *An Introduction to R*. URL http://CRAN.R-project.org/manuals.html. ISBN 3-901167-55-2. 26

Uwe Ligges
Fachbereich Statistik, Universität Dortmund, Germany
ligges@statistik.uni-dortmund.de

Book Reviews

Michael Crawley: Statistical Computing: An Introduction to Data Analysis Using S-Plus

John Wiley and Sons, New York, USA, 2002
770 pages, ISBN 0-471-56040-5
http://www.bio.ic.ac.uk/research/mjcraw/statcomp/

The number of monographs related to the S programming language is sufficiently small (a few dozen, at most) that there are still great opportunities for new books to fill some voids in the literature. *Statistical Computing* (hereafter referred to as *SC*) is one such text, presenting an array of modern statistical methods in S-Plus at an introductory level.

The approach taken by author is to emphasize "graphical data inspection, parameter estimation and model criticism rather than classical hypothesis testing" (p. ix). Within this framework, the text covers a vast range of applied data analysis topics. Background material on S-Plus, probability, and distributions is included in the text. The traditional topics such as linear models, regression, and analysis of variance are covered, sometimes to a greater extent than is common (for example, a split-split-split-split plot experiment and a whole chapter on ANCOVA). More advanced modeling techniques such as bootstrap, generalized linear models, mixed effects models and spatial statistics are also presented. For each topic, some elementary theory is presented and usually an example is worked by hand. Crawley also discusses more philosophical issues such as randomization, replication, model parsimony, appropriate transformations, etc. The book contains a very thorough index of more than 25 pages. In the index, all S-Plus language words appear in bold type.

A supporting web site contains all the data files, scripts with all the code from each chapter of the book, and a few additional (rough) chapters. The scripts pre-suppose that the data sets have already been imported into S-Plus by some mechanism–no explanation of how to do this is given–presenting the novice S-Plus user with a minor obstacle. Once the data is loaded, the scripts can be used to re-do the analyses in the book. Nearly every worked example begins by `attaching` a data frame, and then working with the variable names of the data frame. No `detach` command is present in the scripts, so the work environment can become cluttered and even cause errors if multiple data frames have common column names. Avoid this problem by quitting the S-Plus (or R) session after working through the script for each chapter.

SC contains a very broad use of S-Plus commands to the point the author says about `subplot`, "Having gone to all this trouble, I can't actually see why you would ever want to do this in earnest" (p. 158). Chapter 2 presents a number of functions (`grep`, `regexpr`, `solve`, `sapply`) that are not used within the text, but are usefully included for reference.

Readers of this newsletter will undoubtedly want to know how well the book can be used with R (version 1.6.2 base with recommended packages). According to the preface of *SC*, "The computing is presented in S-Plus, but all the examples will also work in the freeware program called R." Unfortunately, this claim is too strong. The book's web site does contain (as of March, 2003) three modifications for using R with the book. Many more modifications are needed. A rough estimate would be that 90% of the S-Plus code in the scripts needs no modification to work in R. Basic calculations and data manipulations are nearly identical in S-Plus and R. In other cases, a slight change is necessary (`ks.gof` in S-Plus, `ks.test` in R), alternate functions must be used (`multicomp` in S-Plus, `TukeyHSD` in R), and/or additional packages (`nlme`) must be loaded into R. Other topics (bootstrapping, trees) will require more extensive code modification efforts and some functionality is not available in R (`varcomp`, `subplot`).

Because of these differences and the minimal documentation for R within the text, users inexperienced with an S language are likely to find the incompatibilities confusing, especially for self-study of the text. Instructors wishing to use the text with R for teaching should not find it too difficult to help students with the differences in the dialects of S. A complete set of online complements for using R with *SC* would be a useful addition to the book's web site.

In summary, *SC* is a nice addition to the literature of S. The breadth of topics is similar to that of *Modern Applied Statistics with S* (Venables and Ripley, Springer, 2002), but at a more introductory level. With some modification, the book can be used with R. The book is likely to be especially useful as an introduction to a wide variety of data analysis techniques.

Kevin Wright
Pioneer Hi-Bred International
Kevin.Wright@pioneer.com

Peter Dalgaard: Introductory Statistics with R

Springer, Heidelberg, Germany, 2002
282 pages, ISBN 0-387-95475-9
http://www.biostat.ku.dk/~pd/ISwR.html

This is the first book, other than the manuals that are shipped with the R distribution, that is solely devoted to R (and its use for statistical analysis). The author would be familiar to the subscribers of the R mailing lists: Prof. Dalgaard is a member of R Core, and has been providing people on the lists with insightful, elegant solutions to their problems. The author's R wizardry is demonstrated in the many valuable R tips and tricks sprinkled throughout the book, although care is taken to not use too-clever constructs that are likely to leave novices bewildered.

As R is itself originally written as a teaching tool, and many are using it as such (including the author), I am sure its publication is to be welcomed by many. The targeted audience of the book is "nonstatistician scientists in various fields and students of statistics". It is based upon a set of notes the author developed for the course Basic Statistics for Health Researchers at the University of Copenhagen. A couple of caveats were stated in the Preface: Given the origin of the book, the examples used in the book were mostly, if not all, from health sciences. The author also made it clear that the book is not intended to be used as a stand alone text. For teaching, another standard introductory text should be used as a supplement, as many key concepts and definitions are only briefly described. For scientists who have had some statistics, however, this book itself may be sufficient for self-study.

The titles of the chapters are:

1. Basics
2. Probability and distributions
3. Descriptive statistics and graphics
4. One- and two-sample tests
5. Regression and correlation
6. ANOVA and Kruskal-Wallis
7. Tabular data
8. Power and computation of sample sizes
9. Multiple regression
10. Linear Models
11. Logistic regression
12. Survival analysis

plus three appendices: Obtaining and installing R, Data sets in the `ISwR` package (help files for data sets the package), and Compendium (short summary of the R language). Each chapter ends with a few exercises involving further analysis of example data. When I first opened the book, I was a bit surprised to see the last four chapters. I don't think most people expected to see those topics covered in an introductory text. I do agree with the author that these topics are quite essential for data analysis tasks for practical research.

Chapter one covers the basics of the R language, as much as needed in interactive use for most data analysis tasks. People with some experience with S-PLUS but new to R will soon appreciate many nuggets of features that make R easier to use (e.g., some vectorized graphical parameters such as `col`, `pch`, etc.; the functions `subset`, `transform`, among others). As much programming as is typically needed for interactive use (and may be a little more; e.g., the use of `deparse(substitute(...)` as default argument for the title of a plot) is also described.

Chapter two covers the calculation of probability distributions and random number generation for simulations in R. Chapter three describes descriptive statistics (mean, SD, quantiles, summary tables, etc.), as well as how to graphically display summary plots such as Q-Q plots, empirical CDFs, boxplots, dotcharts, bar plots, etc. Chapter four describes the *t* and Wilcoxon tests for one- and two-sample comparison as well as paired data, and the *F* test for equality of variances.

Chapter five covers simple linear regression and bivariate correlations. Chapter six contains quite a bit of material: from oneway ANOVA, pairwise comparisons, Welch's modified F test, Barlett's test for equal variances, to twoway ANOVA and even a repeated measures example. Nonparametric procedures are also described. Chapter seven covers basic inferences for categorical data (tests for proportions, contingency tables, etc.). Chapter eight covers power and sample size calculations for means and proportions in one- and two-sample problems.

The last four chapters provide a very nice, brief summary on the more advanced topics. The graphical presentations of regression diagnostics (e.g., color coding the magnitudes of the diagnostic measures) can be very useful. However, I would have preferred the omission of coverage on variable selection in multiple linear regression entirely. The coverage on these topics is necessarily brief, but do provide enough material for the reader to follow through the examples. It is impressive to cover this many topics in such brevity, without sacrificing clarity of the descriptions. (The linear models chapter covers polynomial regression, ANCOVA, regression diagnostics, among others.)

As with (almost?) all first editions, typos are inevitable (e.g., on page 33, in the modification to vectorize Newton's method for calculating square roots, `any` should have been `all`; on page 191, the logit should be defined as $\log[p/(1-p)]$). On the whole, however, this is a very well written book, with logical ordering of the content and a nice flow. Ample code examples are interspersed throughout the text, making it easy for the reader to follow along on a computer. I whole-heartedly recommend this book to people in the "intended audience" population.

Andy Liaw
Merck Research Laboratories
andy_liaw@merck.com

John Fox: An R and S-Plus Companion to Applied Regression

Sage Publications, Thousand Oaks, CA, USA, 2002
328 pages, ISBN 0-7619-2280-6
http://www.socsci.mcmaster.ca/jfox/Books/companion/

The aim of this book is to enable people with knowledge on linear regression modelling to analyze their data using R or S-Plus. Any textbook on linear regression may provide the necessary background, of course both language and contents of this companion are closely aligned with the authors own *Applied Regression, Linear Models and Related Metods* (Fox, Sage Publications, 1997).

Chapters 1–2 give a basic introduction to R and S-Plus, including a thorough treatment of data import/export and handling of missing values. Chapter 3 complements the introduction with exploratory data analysis and Box-Cox data transformations. The target audience of the first three sections are clearly new users who might never have used R or S-Plus before.

Chapters 4–6 form the main part of the book and show how to apply linear models, generalized linear models and regression diagnostics, respectively. This includes detailed explanations of model formulae, dummy-variable regression and contrasts for categorical data, and interaction effects. Analysis of variance is mostly explained using "Type II" tests as implemented by the `Anova()` function in package **car**. The infamous "Type III" tests, which are requested frequently on the `r-help` mailing list, are provided by the same function, but their usage is discouraged quite strongly in both book and help page.

Chapter 5 reads as a natural extension of linear models and concentrates on error distributions provided by the exponential family and corresponding link functions. The main focus is on models for categorical responses and count data. Much of the functionality provided by package **car** is on regression diagnostics and has methods both for linear and generalized linear models. Heavy usage of graphics, `identify()` and text-based menus for plots emphasize the interactive and exploratory side of regression modelling. Other sections deal with Box-Cox transformations, detection of non-linearities and variable selection.

Finally, Chapters 7 & 8 extend the introductory Chapters 1–3 with material on drawing graphics and S programming. Although the examples use regression models, e.g., plotting the surface of fitted values of a generalized linear model using `persp()`, both chapters are rather general and not "regression-only".

Several real world data sets are used throughout the book. All S code necessary to reproduce numerical and graphical results is shown and explained in detail. Package **car**, which contains all data sets used in the book and functions for ANOVA and regression diagnostics, is available from CRAN and the book's homepage.

The style of presentation is rather informal and hands-on, software usage is demonstrated using examples, without lengthy discussions of theory. Over wide parts the text can be directly used in class to demonstrate regression modelling with S to students (after more theoretical lectures on the topic). However, the book contains enough "reminders" about the theory, such that it is not necessary to switch to a more theoretical text while reading it.

The book is self-contained with respect to S and should easily get new users started. The placement of R *before* S-Plus in the title is not only lexicographic, the author uses R as the primary S engine. Differences of S-Plus (as compared with R) are clearly marked in framed boxes, the graphical user interface of S-Plus is not covered at all.

The only drawback of the book follows directly from the target audience of possible S novices: More experienced S users will probably be disappointed by the ratio of S introduction to material on regression analysis. Only about 130 out of 300 pages deal directly with linear models, the major part is an introduction to S. The online appendix at the book's homepage (mirrored on CRAN) contains several extension chapters on more advanced topics like bootstrapping, time series, nonlinear or robust regression.

In summary, I highly recommend the book to anyone who wants to learn or teach applied regression analysis with S.

Friedrich Leisch
Universität Wien, Austria
Friedrich.Leisch@R-project.org

Changes in R 1.7.0

by the R Core Team

User-visible changes

- `solve()`, `chol()`, `eigen()` and `svd()` now use LAPACK routines unless a new back-compatibility option is turned on. The signs and normalization of eigen/singular vectors may change from earlier versions.

- The 'methods', 'modreg', 'mva', 'nls' and 'ts' packages are now attached by default at startup (in addition to 'ctest'). The option `"defaultPackages"` has been added which contains the initial list of packages. See `?Startup` and `?options` for details. Note that `.First()` is no longer used by R itself.

  `class()` now always (not just when 'methods' is attached) gives a non-null class, and `UseMethod()` always dispatches on the class that `class()` returns. This means that methods like foo.matrix and foo.integer will be used. Functions `oldClass()` and `oldClass<-()` get and set the "class" attribute as R without 'methods' used to.

- The default random number generators have been changed to 'Mersenne-Twister' and 'Inversion'. A new `RNGversion()` function allows you to restore the generators of an earlier R version if reproducibility is required.

- Namespaces can now be defined for packages other than 'base': see 'Writing R Extensions'. This hides some internal objects and changes the search path from objects in a namespace. All the base packages (except methods and tcltk) have namespaces, as well as the recommended packages 'KernSmooth', 'MASS', 'boot', 'class', 'nnet', 'rpart' and 'spatial'.

- Formulae are not longer automatically simplified when `terms()` is called, so the formulae in results may still be in the original form rather than the equivalent simplified form (which may have reordered the terms): the results are now much closer to those of S.

- The tables for plotmath, Hershey and Japanese have been moved from the help pages (`example(plotmath)` etc) to `demo(plotmath)` etc.

- Errors and warnings are sent to stderr not stdout on command-line versions of R (Unix and Windows).

- The R_X11 module is no longer loaded until it is needed, so do test that `x11()` works in a new Unix-alike R installation.

New features

- `if()` and `while()` give a warning if called with a vector condition.

- Installed packages under Unix without compiled code are no longer stamped with the platform and can be copied to other Unix-alike platforms (but not to other OSes because of potential problems with line endings and OS-specific help files).

- The internal random number generators will now never return values of 0 or 1 for `runif()`. This might affect simulation output in extremely rare cases. Note that this is not guaranteed for user-supplied random-number generators, nor when the standalone Rmath library is used.

- When assigning names to a vector, a value that is too short is padded by character NAs. (Wishlist part of PR#2358)

- It is now recommended to use the 'SystemRequirements:' field in the DESCRIPTION file for specifying dependencies external to the R system.

- Output text connections no longer have a line-length limit.

- On platforms where vsnprintf does not return the needed buffer size the output line-length limit for `fifo()`, `gzfile()` and `bzfile()` has been raised from 10k to 100k chars.

- The Math group generic does not check the number of arguments supplied before dispatch: it used to if the default method had one argument but not if it had two. This allows `trunc.POSIXt()` to be called via the group generic `trunc()`.

- Logical matrix replacement indexing of data frames is now implemented (interpreted as if the lhs was a matrix).

- Recursive indexing of lists is allowed, so `x[[c(4,2)]]` is shorthand for `x[[4]][[2]]` etc. (Wishlist PR#1588)

- Most of the time series functions now check explicitly for a numeric time series, rather than fail at a later stage.

- The postscript output makes use of relative moves, and so is somewhat more compact.

- `%*%` and `crossprod()` for complex arguments make use of BLAS routines and so may be much faster on some platforms.

- `arima()` has `coef()`, `logLik()` (and hence AIC) and `vcov()` methods.

- New function `as.difftime()` for time-interval data.

- `basename()` and `dirname()` are now vectorized.

- `biplot.default()` mva allows 'xlab' and 'ylab' parameters to be set (without partially matching to 'xlabs' and 'ylabs'). (Thanks to Uwe Ligges.)

- New function `capture.output()` to send printed output from an expression to a connection or a text string.

- `ccf()` (pckage ts) now coerces its `x` and `y` arguments to class `"ts"`.

- `chol()` and `chol2inv()` now use LAPACK routines by default.

- `as.dist(.)` is now idempotent, i.e., works for `"dist"` objects.

- Generic function `confint()` and 'lm' method (formerly in package MASS, which has 'glm' and 'nls' methods).

- New function `constrOptim()` for optimisation under linear inequality constraints.

- Add 'difftime' subscript method and methods for the group generics. (Thereby fixing PR#2345)

- `download.file()` can now use HTTP proxies which require 'basic' username/password authentication.

- `dump()` has a new argument 'envir'. The search for named objects now starts by default in the environment from which `dump()` is called.

- The `edit.matrix()` and `edit.data.frame()` editors can now handle logical data.

- New argument 'local' for `example()` (suggested by Andy Liaw).

- New function `file.symlink()` to create symbolic file links where supported by the OS.

- New generic function `flush()` with a method to flush connections.

- New function `force()` to force evaluation of a formal argument.

- New functions `getFromNamespace()`, `fixInNamespace()` and `getS3method()` to facilitate developing code in packages with namespaces.

- `glm()` now accepts 'etastart' and 'mustart' as alternative ways to express starting values.

- New function `gzcon()` which wraps a connection and provides (de)compression compatible with gzip.

  `load()` now uses `gzcon()`, so can read compressed saves from suitable connections.

- `help.search()` can now reliably match individual aliases and keywords, provided that all packages searched were installed using R 1.7.0 or newer.

- `hist.default()` now returns the nominal break points, not those adjusted for numerical tolerances.

  To guard against unthinking use, 'include.lowest' in `hist.default()` is now ignored, with a warning, unless 'breaks' is a vector. (It either generated an error or had no effect, depending how prettification of the range operated.)

- New generic functions `influence()`, `hatvalues()` and `dfbeta()` with lm and glm methods; the previously normal functions `rstudent()`, `rstandard()`, `cooks.distance()` and `dfbetas()` became generic. These have changed behavior for glm objects – all originating from John Fox' car package.

- `interaction.plot()` has several new arguments, and the legend is not clipped anymore by default. It internally uses `axis(1,*)` instead of `mtext()`. This also addresses "bugs" PR#820, PR#1305, PR#1899.
- New `isoreg()` function and class for isotonic regression ('modreg' package).
- `La.chol()` and `La.chol2inv()` now give interpretable error messages rather than LAPACK error codes.
- `legend()` has a new 'plot' argument. Setting it 'FALSE' gives size information without plotting (suggested by U.Ligges).
- `library()` was changed so that when the methods package is attached it no longer complains about formal generic functions not specific to the library.
- `list.files()`/`dir()` have a new argument 'recursive'.
- `lm.influence()` has a new 'do.coef' argument allowing *not* to compute casewise changed coefficients. This makes `plot.lm()` much quicker for large data sets.
- `load()` now returns invisibly a character vector of the names of the objects which were restored.
- New convenience function `loadURL()` to allow loading data files from URLs (requested by Frank Harrell).
- New function `mapply()`, a multivariate `lapply()`.
- New function `md5sum()` in package tools to calculate MD5 checksums on files (e.g. on parts of the R installation).
- `medpolish()` package eda now has an 'na.rm' argument (PR#2298).
- `methods()` now looks for registered methods in namespaces, and knows about many objects that look like methods but are not.
- `mosaicplot()` has a new default for 'main', and supports the 'las' argument (contributed by Uwe Ligges and Wolfram Fischer).
- An attempt to `open()` an already open connection will be detected and ignored with a warning. This avoids improperly closing some types of connections if they are opened repeatedly.
- `optim(method = "SANN")` can now cover combinatorial optimization by supplying a move function as the 'gr' argument (contributed by Adrian Trapletti).
- PDF files produced by `pdf()` have more extensive information fields, including the version of R that produced them.
- On Unix(-alike) systems the default PDF viewer is now determined during configuration, and available as the 'pdfviewer' option.
- `pie(...)` has always accepted graphical pars but only passed them on to `title()`. Now `pie(, cex=1.5)` works.
- plot.dendrogram ('mva' package) now draws leaf labels if present by default.
- New `plot.design()` function as in S.
- The `postscript()` and `PDF()` drivers now allow the title to be set.
- New function `power.anova.test()`, contributed by Claus Ekstrøm.
- `power.t.test()` now behaves correctly for negative delta in the two-tailed case.
- `power.t.test()` and `power.prop.test()` now have a 'strict' argument that includes rejections in the "wrong tail" in the power calculation. (Based in part on code suggested by Ulrich Halekoh.)
- `prcomp()` is now fast for nm inputs with $m >> n$.
- `princomp()` no longer allows the use of more variables than units: use `prcomp()` instead.
- `princomp.formula()` now has principal argument 'formula', so `update()` can be used.
- Printing an object with attributes now dispatches on the `class(es)` of the attributes. See `?print.default` for the fine print. (PR#2506)
- `print.matrix()` and `prmatrix()` are now separate functions. `prmatrix()` is the old S-compatible function, and `print.matrix()` is a proper print method, currently identical to `print.default()`. `prmatrix()` and the old `print.matrix()` did not print attributes of a matrix, but the new `print.matrix()` does.
- `print.summary.lm()` and `print.summary.glm()` now default to `symbolic.cor = FALSE`, but `symbolic.cor` can be passed to the print methods from the summary methods. `print.summary.lm()` and `print.summary.glm()` print correlations to 2 decimal places, and the symbolic printout avoids abbreviating labels.

- If a `prompt()` method is called with 'filename' as 'NA', a list-style representation of the documentation shell generated is returned. New function `promptData()` for documenting objects as data sets.

- `qqnorm()` and `qqline()` have an optional logical argument 'datax' to transpose the plot (S-PLUS compatibility).

- `qr()` now has the option to use LAPACK routines, and the results can be used by the helper routines `qr.coef()`, `qr.qy()` and `qr.qty()`. The LAPACK-using versions may be much faster for large matrices (using an optimized BLAS) but are less flexible.

- QR objects now have class "qr", and `solve.qr()` is now just the method for `solve()` for the class.

- New function `r2dtable()` for generating random samples of two-way tables with given marginals using Patefield's algorithm.

- `rchisq()` now has a non-centrality parameter 'ncp', and there's a C API for `rnchisq()`.

- New generic function `reorder()` with a dendrogram method; new `order.dendrogram()` and `heatmap()`.

- `require()` has a new argument named `character.only` to make it align with library.

- New functions `rmultinom()` and `dmultinom()`, the first one with a C API.

- New function `runmed()` for fast runnning medians ('modreg' package).

- New function `slice.index()` for identifying indexes with respect to slices of an array.

- `solve.default(a)` now gives the dimnames one would expect.

- `stepfun()` has a new 'right' argument for right-continuous step function construction.

- `str()` now shows ordered factors different from unordered ones. It also differentiates "NA" and `as.character(NA)`, also for factor levels.

- `symnum()` has a new logical argument 'abbr.colnames'.

- `summary(<logical>)` now mentions NA's as suggested by Göran Broström.

- `summaryRprof()` now prints times with a precision appropriate to the sampling interval, rather than always to 2dp.

- New function `Sys.getpid()` to get the process ID of the R session.

- `table()` now allows exclude= with factor arguments (requested by Michael Friendly).

- The `tempfile()` function now takes an optional second argument giving the directory name.

- The ordering of terms for

  ```
  terms.formula(keep.order=FALSE)
  ```

  is now defined on the help page and used consistently, so that repeated calls will not alter the ordering (which is why `delete.response()` was failing: see the bug fixes). The formula is not simplified unless the new argument 'simplify' is true.

- added "[" method for terms objects.

- New argument 'silent' to `try()`.

- `ts()` now allows arbitrary values for `y` in `start/end = c(x, y)`: it always allowed `y < 1` but objected to `y > frequency`.

- `unique.default()` now works for POSIXct objects, and hence so does `factor()`.

- Package tcltk now allows return values from the R side to the Tcl side in callbacks and the R_eval command. If the return value from the R function or expression is of class "tclObj" then it will be returned to Tcl.

- A new HIGHLY EXPERIMENTAL graphical user interface using the tcltk package is provided. Currently, little more than a proof of concept. It can be started by calling "R -g Tk" (this may change in later versions) or by evaluating `tkStartGUI()`. Only Unix-like systems for now. It is not too stable at this point; in particular, signal handling is not working properly.

- Changes to support name spaces:
  - Placing base in a name space can no longer be disabled by defining the environment variable R_NO_BASE_NAMESPACE.
  - New function `topenv()` to determine the nearest top level environment (usually .GlobalEnv or a name space environment).
  - Added name space support for packages that do not use methods.

- Formal classes and methods can be 'sealed', by using the corresponding argument to setClass or setMethod. New functions `isSealedClass()` and `isSealedMethod()` test sealing.

- packages can now be loaded with version numbers. This allows for multiple versions of files to be installed (and potentially loaded). Some serious testing will be going on, but it should have no effect unless specifically asked for.

Installation changes

- TITLE files in packages are no longer used, the Title field in the DESCRIPTION file being preferred. TITLE files will be ignored in both installed packages and source packages.

- When searching for a Fortran 77 compiler, configure by default now also looks for Fujitsu's frt and Compaq's fort, but no longer for cf77 and cft77.

- Configure checks that mixed C/Fortran code can be run before checking compatibility on ints and doubles: the latter test was sometimes failing because the Fortran libraries were not found.

- PCRE and bzip2 are built from versions in the R sources if the appropriate library is not found.

- New configure option '`--with-lapack`' to allow high-performance LAPACK libraries to be used: a generic LAPACK library will be used if found. This option is not the default.

- New configure options '`--with-libpng`', '`--with-jpeglib`', '`--with-zlib`', '`--with-bzlib`' and '`--with-pcre`', principally to allow these libraries to be avoided if they are unsuitable.

- If the precious variable R_BROWSER is set at configure time it overrides the automatic selection of the default browser. It should be set to the full path unless the browser appears at different locations on different client machines.

- Perl requirements are down again to 5.004 or newer.

- Autoconf 2.57 or later is required to build the configure script.

- Configure provides a more comprehensive summary of its results.

- Index generation now happens when installing source packages using R code in package tools. An existing 'INDEX' file is used as is; otherwise, it is automatically generated from the `\name` and `\title` entries in the Rd files. Data, demo and vignette indices are computed from all available files of the respective kind, and the corresponding index information (in the Rd files, the 'demo/00Index' file, and the `\VignetteIndexEntry{}` entries, respectively). These index files, as well as the package Rd contents data base, are serialized as R objects in the 'Meta' subdirectory of the top-level package directory, allowing for faster and more reliable index-based computations (e.g., in `help.search()`).

- The Rd contents data base is now computed when installing source packages using R code in package tools. The information is represented as a data frame without collapsing the aliases and keywords, and serialized as an R object. (The 'CONTENTS' file in Debian Control Format is still written, as it is used by the HTML search engine.)

- A NAMESPACE file in root directory of a source package is copied to the root of the package installation directory. Attempting to install a package with a NAMESPACE file using '`--save`' signals an error; this is a temporary measure.

Deprecated & defunct

- The assignment operator '_' will be removed in the next release and users are now warned on every usage: you may even see multiple warnings for each usage.

  If environment variable R_NO_UNDERLINE is set to anything of positive length then use of '_' becomes a syntax error.

- `machine()`, `Machine()` and `Platform()` are defunct.

- `restart()` is defunct. Use `try()`, as has long been recommended.

- The deprecated arguments 'pkg' and 'lib' of `system.file()` have been removed.

- `printNoClass()` methods is deprecated (and moved to base, since it was a copy of a base function).

- Primitives `dataClass()` and `objWithClass()` have been replaced by `class()` and `class<-()`; they were internal support functions for use by package methods.

- The use of SIGUSR2 to quit a running R process under Unix is deprecated, the signal may need to be reclaimed for other purposes.

Utilities

- R CMD check more compactly displays the tests of DESCRIPTION meta-information. It now reports demos and vignettes without available index information. Unless installation tests are skipped, checking is aborted if the package dependencies cannot be resolved at run time. Rd files are now also explicitly checked for empty `\name` and `\title` entries. The examples are always run with T and F redefined to give an error if used instead of TRUE and FALSE.

- The Perl code to build help now removes an existing example file if there are no examples in the current help file.

- R CMD Rdindex is now deprecated in favor of function `Rdindex()` in package tools.

- `Sweave()` now encloses the Sinput and Soutput environments of each chunk in an Schunk environment. This allows to fix some vertical spacing problems when using the latex class slides.

C-level facilities

- A full double-precision LAPACK shared library is made available as -lRlapack. To use this include `$(LAPACK_LIBS) $(BLAS_LIBS)` in PKG_LIBS.

- Header file R_ext/Lapack.h added. C declarations of BLAS routines moved to R_ext/BLAS.h and included in R_ext/Applic.h and R_ext/Linpack.h for backward compatibility.

- R will automatically call initialization and unload routines, if present, in shared libraries/DLLs during `dyn.load()` and `dyn.unload()` calls. The routines are named R_init_<dll name> and R_unload_<dll name>, respectively. See the Writing R Extensions Manual for more information.

- Routines exported directly from the R executable for use with `.C()`, `.Call()`, `.Fortran()` and `.External()` are now accessed via the registration mechanism (optionally) used by packages. The ROUTINES file (in src/appl/) and associated scripts to generate FFTab.h and FFDecl.h are no longer used.

- Entry point Rf_append is no longer in the installed headers (but is still available). It is apparently unused.

- Many conflicts between other headers and R's can be avoided by defining STRICT_R_HEADERS and/or R_NO_REMAP – see 'Writing R Extensions' for details.

- New entry point R_GetX11Image and formerly undocumented ptr_R_GetX11Image are in new header R_ext/GetX11Image. These are used by package tkrplot.

Changes on CRAN

by Kurt Hornik and Friedrich Leisch

New contributed packages

Davies useful functions for the Davies quantile function and the Generalized Lambda distribution. By Robin Hankin.

GRASS Interface between GRASS 5.0 geographical information system and R, based on starting R from within the GRASS environment using values of environment variables set in the GISRC file. Interface examples should be run outside GRASS, others may be run within. Wrapper and helper functions are provided for a range of R functions to match the interface metadata structures. Interface functions by Roger Bivand, wrapper and helper functions modified from various originals by interface author.

MCMCpack This package contains functions for posterior simulation for a number of statistical models. All simulation is done in compiled C++ written in the Scythe Statistical Library Version 0.3. All models return coda mcmc objects that can then be summarized using coda functions or the coda menu interface. The package also contains some useful utility functions, including some additional PDFs and pseudo-random number generators for statistical distributions. By Andrew D. Martin, and Kevin M. Quinn.

RSvgDevice A graphics device for R that uses the new w3.org xml standard for Scalable Vector Graphics. By T Jake Luciani.

SenSrivastava Collection of datasets from Sen & Srivastava: Regression Analysis, Theory, Methods and Applications, Springer. Sources for individual data files are more fully documented in the book. By Kjetil Halvorsen.

abind Combine multi-dimensional arrays. This is a generalization of cbind and rbind. Takes a sequence of vectors, matrices, or arrays and produces a single array of the same or higher dimension. By Tony Plate and Richard Heiberger..

ade4 Multivariate data analysis and graphical display. By Jean Thioulouse, Anne-Beatrice Dufour and Daniel Chessel.

amap Hierarchical Clustering and Principal Component Analysis (With general and robusts methods). By Antoine Lucas.

anm The package contains an analog model for statistical/empirical downscaling. By Alexandra Imbert & Rasmus E. Benestad.

clim.pact The package contains R functions for retrieving data, making climate analysis and downscaling of monthly mean and daily mean global climate scenarios. By Rasmus E. Benestad.

dispmod Functions for modelling dispersion in GLM. By Luca Scrucca.

effects Graphical and tabular effect displays, e.g., of interactions, for linear and generalised linear models. By John Fox.

gbm This package implements extensions to Freund and Schapire's AdaBoost algorithm and J. Friedman's gradient boosting machine. Includes regression methods for least squares, absolute loss, logistic, Poisson, Cox proportional hazards partial likelihood, and AdaBoost exponential loss. By Greg Ridgeway.

genetics Classes and methods for handling genetic data. Includes classes to represent genotypes and haplotypes at single markers up to multiple markers on multiple chromosomes. Function include allele frequencies, flagging homo/heterozygotes, flagging carriers of certain alleles, computing disequilibrium, testing Hardy-Weinberg equilibrium, ... By Gregory Warnes and Friedrich Leisch.

glmmML A Maximum Likelihood approach to mixed models. By Göran Broström..

gpclib General polygon clipping routines for R based on Alan Murta's C library. By R interface by Roger D. Peng; GPC library by Alan Murta.

grasper R-GRASP uses generalized regressions analyses to automate the production of spatial predictions. By A. Lehmann, J.R. Leathwick, J.McC. Overton, ported to R by F. Fivaz.

gstat variogram modelling; simple, ordinary and universal point or block (co)kriging, and sequential Gaussian or indicator (co)simulation; variogram and map plotting utility functions. By Edzer J. Pebesma and others.

hier.part Variance partition of a multivariate data set. By Chris Walsh and Ralph MacNally.

hwde Fits models for genotypic disequilibria, as described in Huttley and Wilson (2000), Weir (1996) and Weir and Wilson (1986). Contrast terms are available that account for first order interactions between loci. By J.H. Maindonald.

ismev Functions to support the computations carried out in 'An Introduction to Statistical Modeling of Extreme Values' by Stuart Coles. The functions may be divided into the following groups; maxima/minima, order statistics, peaks over thresholds and point processes. Original S functions by Stuart Coles, R port and R documentation files by Alec Stephenson.

locfit Local Regression, Likelihood and density estimation. By Catherine Loader.

mimR An R interface to MIM for graphical modelling in R. By Søren Højsgaard.

mix Estimation/multiple imputation programs for mixed categorical and continuous data . Original by Joseph L. Schafer, R port by Brian Ripley.

multidim multidimensional descritptive statistics: factorial methods and classification. Original by André Carlier & Alain Croquette, R port by Mathieu Ros & Antoine Lucas.

normalp A collection of utilities refered to normal of order p distributions (General Error Distributions). By Angelo M. Mineo.

pls.pcr Multivariate regression by PLS and PCR. By Ron Wehrens.

polspline Routines for the polynomial spline fitting routines hazard regression, hazard estimation with flexible tails, logspline, lspec, polyclass, and polymars, by C. Kooperberg and coauthors . By Charles Kooperberg.

rimage This package provides functions for image processing, including sobel filter, rank filters, fft, histogram equalization, and reading JPEG file. This package requires fftw `http://www.fftw.org/` and libjpeg `http://www.ijg.org>`. By Tomomi Takashina.

session Utility functions for interacting with R processes from external programs. This package includes functions to save and restore session information (including loaded packages, and attached data objects), as well as functions to evaluate strings containing R commands and return the printed results or an execution transcript. By Gregory R. Warnes.

snow Support for simple parallel computing in R. By Luke Tierney, A. J. Rossini, and Na Li.

statmod Miscellaneous biostatistical modelling functions. By Gordon Smyth.

survey Summary statistics, generalised linear models, and general maximum likelihood estimation for stratified, cluster-sampled, unequally weighted survey samples. By Thomas Lumley.

vcd Functions and data sets based on the book "Visualizing Categorical Data" by Michael Friendly. By David Meyer, Achim Zeileis, Alexandros Karatzoglou, and Kurt Hornik.

New Omegahat packages

REventLoop An abstract event loop mechanism that is toolkit independent and can be used to to replace the R event loop. This allows one to use the Gtk or Tcl/Tk's event loop which gives better response and coverage of all event sources (e.g. idle events, timers, etc.) . By Duncan Temple Lang.

RGdkPixbuf S language functions to access the facilities in GdkPixbuf for manipulating images. This is used for loading icons to be used in widgets such as buttons, HTML renderers, etc. By Duncan Temple Lang.

RGtkExtra A collection of S functions that provide an interface to the widgets in the gtk+extra library such as the GtkSheet data-grid display, icon list, file list and directory tree. By Duncan Temple Lang.

RGtkGlade S language bindings providing an interface to Glade, the interactive Gnome GUI creator. This allows one to instantiate GUIs built with the interactive Glade tool directly within S. The callbacks can be specified as S functions within the GUI builder. By Duncan Temple Lang.

RGtkHTML A collection of S functions that provide an interface to creating and controlling an HTML widget which can be used to display HTML documents from files or content generated dynamically in S. This also supports embedded objects created and managed within S code, include graphics devices, etc. By Duncan Temple Lang.

RGtk Facilities in the S language for programming graphical interfaces using Gtk, the Gnome GUI toolkit. By Duncan Temple Lang.

SWinRegistry Provides access from within R to read and write the Windows registry. By Duncan Temple Lang.

SWinTypeLibs This provides ways to extract type information from type libraries and/or DCOM objects that describes the methods, properties, etc. of an interface. This can be used to discover available facilities in a COM object or automatically generate S and C code to interface to such objects. By Duncan Temple Lang.

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

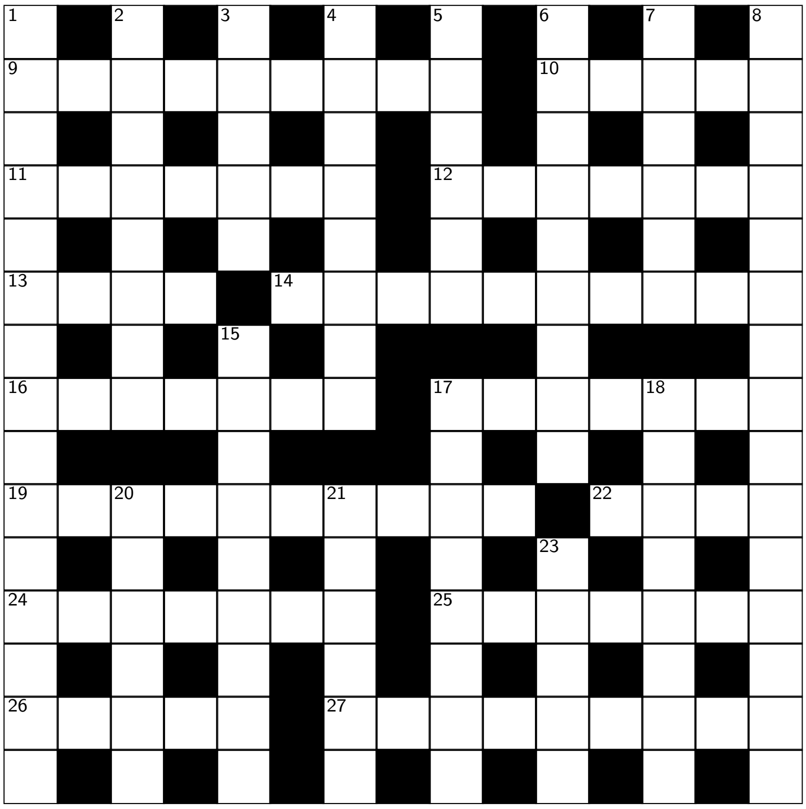
Friedrich Leisch
Technische Universität Wien, Austria
Friedrich.Leisch@R-project.org

Crossword

by Barry Rowlingson

Following a conversation with 1 down at DSC 2003 post-conference dinner, I have constructed a crossword for R-news. Most of the clues are based on matters relating to R or computing in general. The clues are 'cryptic', in that they rely on word-play, but most will contain a definition somewhere. A prize of EUR50 is on offer. See http://www.maths.lancs.ac.uk/~rowlings/Crossword for details.

Barry Rowlingson
Lancaster University, UK
B.Rowlingson@lancaster.ac.uk



Across

9 Possessing a file, or boat, we hear (9)
10, 20 down Developer code is binary peril (5,6)
11 Maturer code has no need for this (7)
12 And a volcano implodes moderately slowly (7)
13 Partly not an hyperbolic function (4)
14 Give 500 euro to R Foundation, then stir your café Breton (10)
16 Catches crooked parents (7)
17, 23 down Developer code bloated Gauss (7,5)
19 Fast header code is re-entrant (6-4)
22, 4 down Developer makes toilet compartments (4,8)
24 Former docks ship files (7)
25 Analysis of a skinned banana and eggs (2,5)
26 Rearrange array or a hairstyle (5)
27 Atomic number 13 is a component (9)

Down

1 Scots king goes with lady president (6,9)
2 Assemble rain tent for private space (8)
3 Output sounds sound (5)
4 See 22ac
5 Climb tree to get new software (6)
6 Naughty number could be 8 down (1,3,5)
7 Failed or applied ! (3,3)
8 International body member is ordered back before baker, but can't be shown (15)
15 I hear you consort with a skeleton, having rows (9)
17 Left dead encrypted file in zip archive (8)
18 Guards riot, loot souk (8)
20 See 10ac
21 A computer sets my puzzle (6)
23 See 17ac

Recent Events

DSC 2003

Inspired by the success of its predecessors in 1999 and 2001, the Austrian Association for Statistical Computing and the R Foundation for Statistical Computing jointly organised the third workshop on distributed statistical computing which took place at the Technische Universität Vienna from March 20 to March 22, 2003. More than 150 participants from all over the world enjoyed this most interesting event. The talks presented at DSC 2003 covered a wide range of topics from bioinformatics, visualisation, resampling and combine methods, spatial statistics, user-interfaces and office integration, graphs and graphical models, database connectivity and applications. Compared to DSC 2001, a greater fraction of talks addressed problems not directly related to R, however, most speakers mentioned R as their favourite computing engine. An online proceedings volume is currently under preparation.

Two events took place in Vienna just prior to DSC 2003. Future directions of R developments were discussed at a two-day open R-core meeting. Tutorials on the Bioconductor project, R programming and R graphics attended a lot of interest the day before the conference opening.

While the talks were given in the beautiful rooms of the main building of the Technische Universität at Karlsplatz in the city centre, the fruitful discussions continued in Vienna's beer pubs and at the conference dinner in one of the famous Heurigen restaurants.

Finally, one participant summarised this event most to the point in an email sent to me right after DSC 2003: "At most statistics conferences, over 90% of the stuff is not interesting to me, at this one, over 90% of it was interesting."

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

R News is a 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, all other submissions to the editor-in-chief or another member of the editorial board (more detailed submission instructions can be found on the R homepage).

R Project Homepage:
http://www.R-project.org/

This newsletter is available online at
http://CRAN.R-project.org/doc/Rnews/