Extending the R Commander by "Plug-In" Packages

by John Fox

This article describes how the **Rcmdr** package can be extended by suitably designed "plug-in" packages. Embodied in the **Rcmdr** package, the **R Commander** was originally conceived as a basic-statistics graphical user interface ("GUI") to **R**. The **R Commander**'s capabilities have since been extended substantially beyond this original purpose, although it still accesses only a small fraction of the statistical and data-management capabilities of the standard **R** distribution, not to mention those of the more than 1000 contributed packages now on CRAN.

In addition to enhancing the usefulness of the **R Commander** as a teaching tool, the plug-in package mechanism allows interested package developers to add graphical interfaces to their **R** software, with the **R Commander** providing most of the necessary intrastructure, and without — as was previously the case — requiring the developer to maintain an independent version of the **Rcmdr** package [e.g., Dusa (2007)].

The **Rcmdr** package is based on the **tcltk** package (Dalgaard, 2001, 2002), which provides an **R** interface to the **Tcl/Tk** GUI builder. Because **Tcl/Tk** and the **tcltk** package are available for all of the computing platforms on which **R** is commonly run, the **R Commander** GUI runs on all of these platforms as well.

The main **R Commander** window is shown in Figure 1. Along the top are the **R Commander** menus: *File*, *Edit*, *Data*, *Statistics*, and so on. Below the menus is a tool bar, containing buttons for selecting, editing, and viewing the "active" data set, and for selecting the "active" statistical model. Below the toolbar are script, output, and messages windows: Commands generated by the **R Commander** appear in the script window; these commands can be edited and re-executed. Printed output is displayed in the output window, while error messages, warnings, and notes appear in the messages window. A more detailed description of the **R Commander** interface may be found in Fox (2005).

Workflow in the **R Commander** is straightforward, and is based on a single rectangular (i.e., case-by-variable) data set being "active" at any given time. Users can import, select, or modify the active data set via the **R Commander**'s *Data* menu. Various statistical methods are provided by the *Statistics* and *Graphs* menus. The **R Commander** recognizes certain classes of objects as statistical models. There can be an active statistical model that is manipulated via the *Models* menu. As long as a package conforms to this simple design (and does not, for example, require access to several data frames simultaneously), it should be possible to integrate it with the **R Commander**.

From a very early stage in its development, the **R Commander** was designed to be extensible: The menus for the **R Commander** interface are not hard-coded in the package sources, but rather are defined in a plain-text configuration file that installs into the **Rcmdr** package's `etc` subdirectory. Similarly, when it starts up, the **R Commander** will automatically source files with `.R` extensions that reside in the **Rcmdr** `etc` subdirectory; such files can contain **R** code to build **R Commander** dialog boxes, for example. Nevertheless, extending the **R Commander** has been relatively inconvenient, requiring either that users modify the installed package (e.g., by editing the **R Commander** menu-configuration file), or that they modify and recompile the **Rcmdr** source package.[1]

Starting with version 1.3-0, the **Rcmdr** makes alternative provision for extension by "plug-in" packages — standard **R** packages that are developed, maintained, distributed, and installed independently of the **Rcmdr** package. Plug-in packages can augment the **R Commander** menus, and can provide additional dialog boxes and statistical functionality. Once installed on the user's system, plug-in packages are automatically detected when the **R Commander** starts up and can be loaded from the **R Commander**'s *Tools* menu. Plug-in packages can alternatively be loaded independently via **R**'s `library` command; in this event, the **Rcmdr** package will also be loaded with the plug-in's menus installed in the **R Commander** menu bar. Finally, plug-in packages can be loaded automatically along with the **Rcmdr** package by setting the **Rcmdr** `plugins` option; for example, the command

```
options(Rcmdr=list(plugins=
  "RcmdrPlugin.TeachingDemos"))
```

causes the **RcmdrPlugin.TeachingDemos** plug-in package (described below) to load when the **R Commander** starts up.

The remainder of this article explains in some detail how to design **R Commander** plug-in packages. I begin with a description of the **R Commander** menu-definition file, because the format of this file is shared by **R Commander** plug-in packages. I then very briefly explain how to write functions for constructing **R Commander** dialog boxes. Finally, I

[1] In at least one instance, this inconvenience led to the distribution of a complete, alternative version of the **Rcmdr** package, Richard Heiberger and Burt Holland's **Rcmdr.HH** package. Exploiting the new facilities for extending the **Rcmdr** described in this article, Professor Heiberger has redesigned **Rcmdr.HH** as an **R Commander** plug-in package (Heiberger with Holland, 2007).

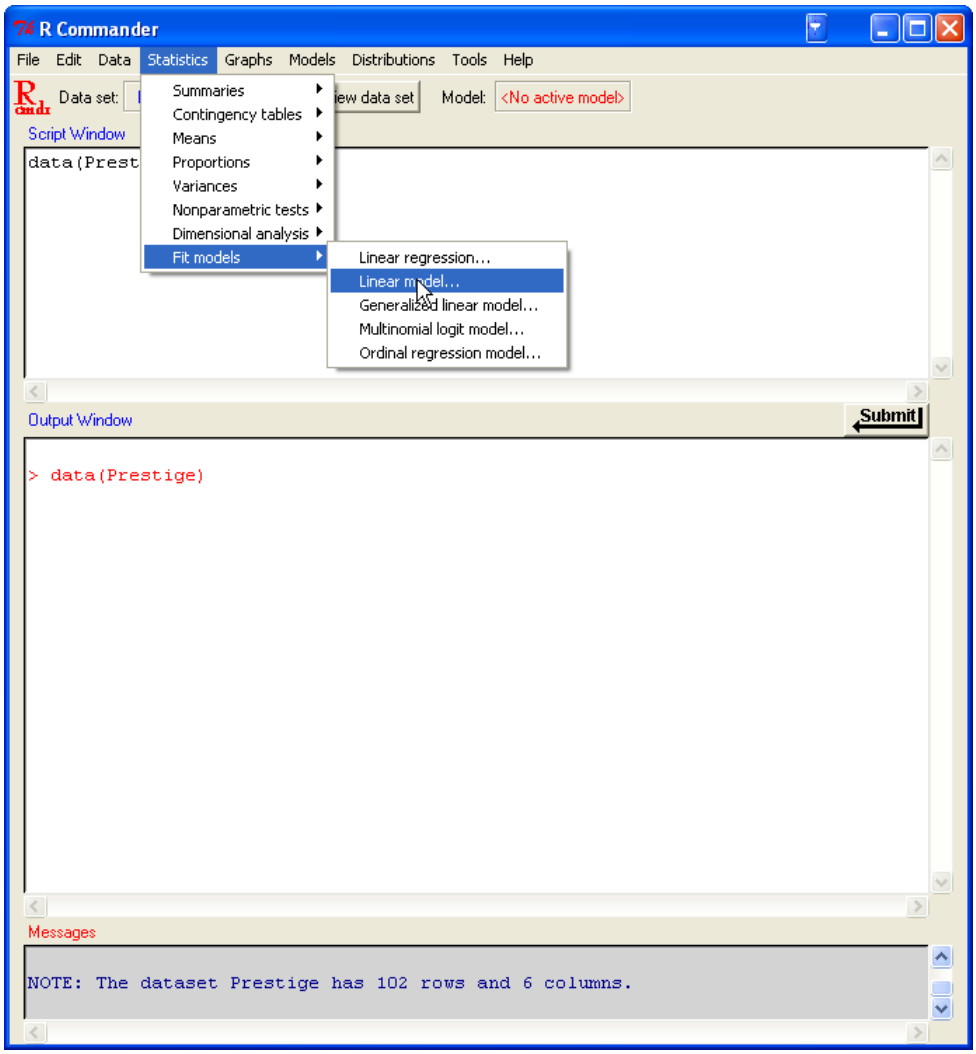

Figure 1: The **R Commander** interface (under **Windows XP**), showing menus, script, output, and messages windows.

describe the structure of plug-in packages.

The Menu-Definition File

The standard **R Commander** menus are defined in the file `Rcmdr-menus.txt`, which resides in the installed **Rcmdr** package's `etc` subdirectory. Each line in the file comprises seven text fields, separated by white space. Fields with embedded blanks must be enclosed in single or double-quotes; unused fields are specified by an empty character string, `""`.

The `Rcmdr-menus.txt` file is quite large, and so I will not reproduce it here (though you may want to look at it in a text editor as you read the following description). Let us instead examine some representative lines in the file. The first line in `Rcmdr-menus.txt` defines the top-level *File* menu:

```
menu fileMenu topMenu  ""  ""  ""  ""
```

- The first, or "operation type", field — `menu` — indicates that we are defining a menu; it is also possible to define a menu item, in which case the operation type is `item` (see below).
- The second field — `fileMenu` — gives an arbitrary name to the new menu; any valid **R** name can be employed.
- The third field — `topMenu` — specifies the "parent" of the menu, in this case signifying that `fileMenu` is a top-level menu, to be installed directly in the **R Commander** menu bar. It is also possible to define a submenu of another menu (see below).
- The remaining four fields are empty.

The second line in `Rcmdr-menus.txt` defines a menu item under `fileMenu`:[2]

```
item fileMenu command "Open script file..."
  loadLog  ""  ""
```

- As explained previously, the first field indicates the definition of a menu item.
- The second field indicates that the menu item belongs to `fileMenu`.
- The third field specifies that the menu item invokes a command.
- The fourth field gives the text corresponding to the menu item that will be displayed when a user opens the *File* menu; the ellipses (`...`) are a conventional indication that selecting this menu item leads to a dialog box (see Figure 2).
- The fifth field specifies the name of the function (`loadLog`) to be called when the menu item is selected. This function is defined in the **Rcmdr** package, but any **R** function that has no required arguments can serve as a menu-item call-back function.
- In this case, the sixth and seventh fields are empty; I will explain their purpose presently.

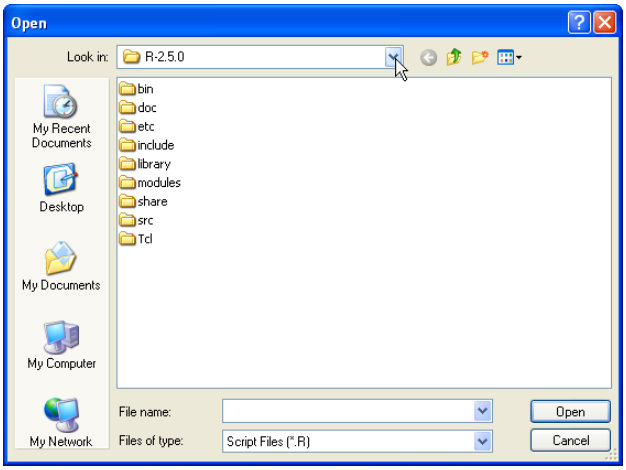

Figure 2: The dialog box produced by the **R Commander** `loadLog` function.

A little later in `Rcmdr-menus.txt`, the following line appears:

```
menu exitMenu fileMenu  ""  ""  ""  ""
```

This line defines a submenu, named `exitMenu`, under `fileMenu`. Subsequent lines (not shown here) define menu items belonging to `exitMenu`.

Still later, we encounter the lines

```
item fileMenu cascade "Exit" exitMenu  ""  ""
item topMenu  cascade "File" fileMenu  ""  ""
```

Each of these lines installs a menu and the items that belong to it, "cascading" the menu under its parent: `exitMenu` is cascaded under `fileMenu`, and will appear with the label *Exit*, while `fileMenu` is installed as a top-level menu with the label *File*. Again, the last two fields are not used.

Fields six and seven control, respectively, the conditional activation and conditional installation of the corresponding item. Each of these fields contains an **R** expression enclosed in quotes that evaluates either to `TRUE` or `FALSE`. Here is an example from `Rcmdr-menus.txt`:

```
item tablesMenu command "Multi-way table..."
  multiWayTable
  "factorsP(3)"  "packageAvailable('abind')"
```

[2] I have broken this — and other — menu lines for purposes of display because they are too long to show on a single line.

This line defines an item under `tablesMenu` (which is a submenu of the *Statistics* menu); the item leads to a dialog box, produced by the call-back function `multiWayTable`, for constructing multi-way contingency tables.

The activation field, `"factorsP(3)"`, returns `TRUE` if the active dataset contains at least three factors — it is, of course, not possible to construct a multi-way table from fewer than three factors. When `factorsP(3)` is `TRUE`, the menu item is activated; otherwise, it is inactive and "grayed out."

The function that constructs multi-way contingency tables requires the `abind` package, both in the sense that it needs this package to operate and in the literal sense that it executes the command `require(abind)`. If the `abind` package is available on the user's system, the command `packageAvailable('abind')` returns `TRUE`. Under these circumstances, the menu item will be installed when the **R Commander** starts up; otherwise, it will not be installed.

Through judicious use of the activation and installation fields, a menu designer, therefore, is able to prevent the user from trying to do some things that are inappropriate in the current context, and even from seeing menu items that cannot work.

R Commander Dialogs

Most **R Commander** menu items lead to dialog boxes. A call-back function producing a dialog can make use of any appropriate **tcltk** commands, and indeed in writing such functions it helps to know something about **Tcl/Tk**. The articles by Dalgaard (2001, 2002) mentioned previously include basic orienting information, and there is a helpful web site of **R tcltk** examples compiled by James Wettenhall, at `http://bioinf.wehi.edu.au/~wettenhall/RTclTkExamples/`. Welch et al. (2003) provide a thorough introduction to **Tcl/Tk**.

In addition, however, the **Rcmdr** package exports a number of functions meant to facilitate the construction of **R Commander** dialogs, and to help insure that these dialogs have a uniform appearance. For example, the **Rcmdr** `radioButtons` function constructs a related set of radio buttons (for selecting one of several choices) in a single simple command (see below). One relatively painless way to proceed, if it is applicable, is to find an existing **R Commander** dialog box that is similar to what you intend to construct and to adapt it.

A reasonably typical, if simple, **Rcmdr** dialog box, for computing a paired *t*-test, is shown in Figure 3. This dialog box is produced by the call-back function `pairedTTest` shown in Figure 4, which illustrates the use of a number of functions exported by the **Rcmdr** package, such as `initializeDialog`, `variableListBox`, and `radioButtons`, as well as some **tlctk** functions that are called directly, such as `tclvalue`, `tkentry`, and `tkgrid`. Additional information about the functions provided by the **Rcmdr** package may be found in Fox (2005) and via `?Rcmdr.Utilities`.

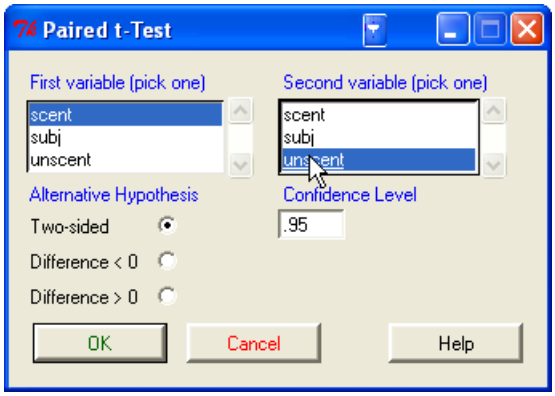

Figure 3: **R Commander** dialog produced by the function `pairedTTest`.

Writing Plug-In Packages

I have contributed an illustrative plug-in package to CRAN, **RcmdrPlugin.TeachingDemos**. The name was selected so that this package will sort alphabetically after the **Rcmdr** package on CRAN; I suggest that other writers of **Rcmdr** plug-ins adopt this naming convention. The **RcmdrPlugin.TeachingDemos** package adds menus to the **R Commander** for some of the demonstrations in Greg Snow's intriguing **TeachingDemos** package (Snow, 2005). In particular, a *Visualize distributions* sub-menu with several items is cascaded under the standard **R Commander** top-level *Distributions* menu, and a new *Demos* top-level menu, also with several menu items, is installed in the **R Commander** menu bar.

An **R Commander** plug-in package is, in the first instance, an ordinary **R** package. Detailed instructions for creating packages are available in the manual *Writing R Extensions* (R Development Core Team, 2007), which ships with **R**.

The `DESCRIPTION` file for the **RcmdrPlugin.TeachingDemos** package is given in Figure 5. All of the fields in this `DESCRIPTION` file are entirely standard, with the exception of the last, `Models:`. Certain classes of objects are recognized by the **R Commander** as statistical models, including objects of class `lm`, `glm`, `multinom`, and `polr`. You can add to this list here, separating the entries by commas, if there are more than one. The **RcmdrPlugin.TeachingDemos** package specifies no additional models; it is, therefore, not necessary to include the `Models:` field in the `DESCRIPTION` file — I have done so simply to indicate its format.

Figure 6 shows the `.First.lib` function for the **RcmdrPlugin.TeachingDemos** package. As is stan-

```
pairedTTest <- function(){
    initializeDialog(title=gettextRcmdr("Paired t-Test"))
    .numeric <- Numeric()
    xBox <- variableListBox(top, .numeric,
        title=gettextRcmdr("First variable (pick one)"))
    yBox <- variableListBox(top, .numeric,
        title=gettextRcmdr("Second variable (pick one)"))
    onOK <- function(){
        x <- getSelection(xBox)
        y <- getSelection(yBox)
        if (length(x) == 0 | length(y) == 0){
            errorCondition(recall=pairedTTest,
                message=gettextRcmdr("You must select two variables."))
            return()
            }
        if (x == y){
            errorCondition(recall=pairedTTest,
                message=gettextRcmdr("Variables must be different."))
            return()
            }
        alternative <- as.character(tclvalue(alternativeVariable))
        level <- tclvalue(confidenceLevel)
        closeDialog()
        .activeDataSet <- ActiveDataSet()
        doItAndPrint(paste("t.test(", .activeDataSet, "$", x, ", ",
            .activeDataSet, "$", y,
            ", alternative='", alternative, "', conf.level=", level,
            ", paired=TRUE)", sep=""))
        tkfocus(CommanderWindow())
        }
    OKCancelHelp(helpSubject="t.test")
    radioButtons(top, name="alternative",
        buttons=c("twosided", "less", "greater"),
        values=c("two.sided", "less", "greater"),
        labels=gettextRcmdr(c("Two-sided", "Difference < 0",
            "Difference > 0")),
        title=gettextRcmdr("Alternative Hypothesis"))
    confidenceFrame <- tkframe(top)
    confidenceLevel <- tclVar(".95")
    confidenceField <- tkentry(confidenceFrame, width="6",
        textvariable=confidenceLevel)
    tkgrid(getFrame(xBox), getFrame(yBox), sticky="nw")
    tkgrid(tklabel(confidenceFrame,
        text=gettextRcmdr("Confidence Level"), fg="blue"))
    tkgrid(confidenceField, sticky="w")
    tkgrid(alternativeFrame, confidenceFrame, sticky="nw")
    tkgrid(buttonsFrame, columnspan=2, sticky="w")
    dialogSuffix(rows=3, columns=2)
    }
```

Figure 4: The `pairedTTest` function.

```
Package: RcmdrPlugin.TeachingDemos
Type: Package
Title: Rcmdr Teaching Demos Plug-In
Version: 1.0-3
Date: 2007-11-02
Author: John Fox <jfox@mcmaster.ca>
Maintainer: John Fox <jfox@mcmaster.ca>
Depends: Rcmdr (>= 1.3-0), rgl, TeachingDemos
Description: This package provides an Rcmdr "plug-in" based on the
  TeachingDemos package, and is primarily for illustrative purposes.
License: GPL (>= 2)
Models:
```

Figure 5: The `DESCRIPTION` file from the `RcmdrPlugin.TeachingDemos` package.

dard in **R**, this function executes when the package is loaded, and serves to load the **Rcmdr** package, with the plug-in activated, if the **Rcmdr** is not already loaded. `.First.lib` is written so that it can (and should) be included in every **R Commander** plug-in package.[3]

Every **R Commander** plug-in package must include a file named `menus.txt`, residing in the installed package's `etc` subdirectory. This file, therefore, should be located in the source package's `inst/etc` subdirectory. A plug-in package's `menus.txt` file has the same structure as `Rcmdr-menus.txt`, described previously. For example, the line

```
menu demosMenu topMenu  ""  ""  ""  ""
```

in the `menus.txt` file for the **RcmdrPlugin.TeachingDemos** package creates a new top-level menu, `demosMenu`;

```
item demosMenu command
  "Central limit theorem..."
  centralLimitTheorem
  ""  "packageAvailable('TeachingDemos')"
```

creates an item under this menu; and

```
item topMenu cascade "Demos" demosMenu
  ""  "packageAvailable('TeachingDemos')"
```

installs the new menu, and its items, in the menu bar (see Figure 7).

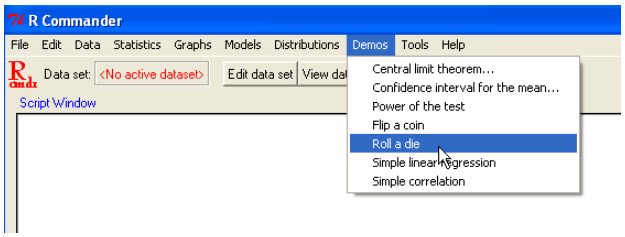

Figure 7: The *Demos* menu provided by the **RcmdrPlugin.TeachingDemos** package.

The **R Commander** takes care of reconciling the `menus.txt` files for plug-in packages with the master `Rcmdr-menus.txt` file: New top-level menus appear to the left of the standard *Tools* and *Help* menus; when new sub-menus or items are inserted into existing menus, they appear at the end. The **RcmdrPlugin.TeachingDemos** package also includes **R** code, for example for the `centralLimitTheorem` call-back function, which creates a dialog box.

Concluding Remarks

It is my hope that the ability to define plug-in packages will extend the utility of the **R Commander** interface. The availability of a variety of specialized plug-ins, and the possibility of writing one's own plug-in package, should allow instructors to tailor the **R Commander** more closely to the specific needs of their classes. Similarly **R** developers wishing to add a GUI to their packages have a convenient means of doing so. An added benefit of having a variety of optionally loaded plug-ins is that unnecessary menus and menu items need not be installed: After all, one of the *disadvantages* of an extensive GUI is that users can easily become lost in a maze of menus and dialogs.

Bibliography

P. Dalgaard. A primer on the R-Tcl/Tk package. *R News*, 1(3):27–31, September 2001. URL `http://CRAN.R-project.org/doc/Rnews/`.

P. Dalgaard. Changes to the R-Tcl/Tk package. *R News*, 2(3):25–27, December 2002. URL `http://CRAN.R-project.org/doc/Rnews/`.

A. Dusa. *QCAGUI: QCA Graphical User Interface*, 2007. R package version 1.3-0.

[3] I am grateful to Richard Heiberger for help in writing this function, and, more generally, for his suggestions for the design of the **Rcmdr** plug-in facility.

```
.First.lib <- function(libname, pkgname){
    if (!interactive()) return()
    Rcmdr <- options()$Rcmdr
    plugins <- Rcmdr$plugins
    if ((!pkgname %in% plugins) && !getRcmdr("autoRestart")) {
        Rcmdr$plugins <- c(plugins, pkgname)
        options(Rcmdr=Rcmdr)
        closeCommander(ask=FALSE, ask.save=TRUE)
        Commander()
        }
    }
```

Figure 6: The `.First.lib` function from the `RcmdrPlugin.TeachingDemos` package.

J. Fox. The R Commander: A basic-statistics graphical user interface to R. *Journal of Statistical Software*, 14(9):1–42, Aug. 2005. ISSN 1548-7660. URL `http://www.jstatsoft.org/counter.php?id=134&url=v14/i09/v14i09.pdf&ct=1`.

R. M. Heiberger and with contributions from Burt Holland. *RcmdrPlugin.HH: Rcmdr support for the HH package*, 2007. R package version 1.1-4.

R Development Core Team. *Writing R Extensions*. 2007.

G. Snow. *TeachingDemos: Demonstrations for teaching and learning*, 2005. R package version 1.5.

B. B. Welch, K. Jones, and J. Hobbs. *Practical Programming in Tcl/Tk, Fourth Edition*. Prentice Hall, Upper Saddle River NJ, 2003.

John Fox
Department of Sociology
McMaster University
Hamilton, Ontario, Canada
`jfox@mcmaster.ca`

Improvements to the Multiple Testing Package multtest

by Sandra L. Taylor, Duncan Temple Lang, and Katherine S. Pollard

Introduction

The R package **multtest** (Dudoit and Ge, 2005) contains multiple testing procedures for analyses of high-dimensional data, such as microarray studies of gene expression. These methods include various marginal p-value adjustment procedures (the `mt.rawp2adjp` function) as well as joint testing procedures. A key component of the joint testing methods is estimation of a null distribution for the vector of test statistics, which is accomplished via permutations (Westfall and Young, 1993; Ge et al., 2003) or the non-parametric bootstrap (Pollard and van der Laan, 2003; Dudoit et al., 2004). Statistical analyses of high-dimensional data often are computationally intensive. Application of resampling-based statistical methods such as bootstrap or permutation methods to these large data sets further increases computational demands. Here we report on improvements incorporated into **multtest** version 1.16.1 available via both *Bioconductor* and *CRAN*. These updates have significantly increased the computational speed of using the bootstrap procedures.

The **multtest** package implements multiple testing procedures with a bootstrap null distribution through the main user function `MTP`. Eight test statistic functions (`meanX`, `diffmeanX`, `FX`, `blockFX`, `twowayFX`, `lmX`, `lmY`, `coxY`) are used to conduct one and two sample *t*-tests, one and two-way ANOVAs, simple linear regressions and survival analyses, respectively. To generate a bootstrap null distribution, the `MTP` function calls the function `boot.null`. This function then calls `boot.resample` to generate bootstrap samples. Thus, the call stack for the bootstrap is `MTP -> boot.null -> boot.resample`. Finally, the test statistic function is applied to each sample, and `boot.null` returns a matrix of centered and scaled bootstrap test statistics.

We increased the computational speed of generating this bootstrap null distribution through three main modifications:

1. optimizing the R code of the test statistics;
2. implementing two frequently used tests (two sample *t*-test and *F*-test) in C; and