

The Journal

Volume 3/2, December 2011

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

Contents

Editorial	3
Contributed Research Articles	
Creating and Deploying an Application with (R)Excel and R	5
glm2: Fitting Generalized Linear Models with Convergence Problems	12
Implementing the Compendium Concept with Sweave and DOCSTRIP	16
Watch Your Spelling!	22
Ckmeans.1d.dp: Optimal k -means Clustering in One Dimension by Dynamic Programming	29
Nonparametric Goodness-of-Fit Tests for Discrete Null Distributions	34
Using the Google Visualisation API with R	40
GrapheR: a Multiplatform GUI for Drawing Customizable Graphs in R	45
rainbow: An R Package for Visualizing Functional Time Series	54
Programmer's Niche	
Portable C++ for R Packages	60
News and Notes	
R's Participation in the Google Summer of Code 2011	64
Conference Report: useR! 2011	68
Forthcoming Events: useR! 2012	70
Changes in R	72
Changes on CRAN	84
News from the Bioconductor Project	86
R Foundation News	87

The  Journal is a peer-reviewed publication of the R Foundation for Statistical Computing. Communications regarding this publication should be addressed to the editors. All articles are licensed under the Creative Commons Attribution 3.0 Unported license (CC BY 3.0, <http://creativecommons.org/licenses/by/3.0/>).

Prospective authors will find detailed and up-to-date submission instructions on the Journal's homepage.

Editor-in-Chief:

Heather Turner
Statistics Department
University of Warwick
Coventry CV4 7AL
UK

Editorial Board:

Peter Dalgaard, Martyn Plummer, and Hadley Wickham.

Editor Programmer's Niche:

Bill Venables

Editor Help Desk:

Uwe Ligges

Editor Book Reviews:

G. Jay Kerns
Department of Mathematics and Statistics
Youngstown State University
Youngstown, Ohio 44555-0002
USA
gkerns@ysu.edu

R Journal Homepage:

<http://journal.r-project.org/>

Email of editors and editorial board:

firstname.lastname@R-project.org

The R Journal is indexed/abstracted by EBSCO, DOAJ.

Editorial

by Heather Turner

Following the guiding principles set down by its predecessor *R News*, the journal has maintained the policy of allowing authors to retain copyright of their published articles. However the lack of an explicit licence has meant that the rights of others regarding material published in the journal has been unclear. Therefore from this issue, all articles will be licensed under the Creative Commons Attribution 3.0 Unported license (CC BY 3.0, <http://creativecommons.org/licenses/by/3.0/>). This license is the standard set by the SPARC Europe Seal for Open Access Journals and ensures compatibility with other open access journals such as the Journal of Statistical Software.

The key features of the CC BY 3.0 license are that anyone is free to share or to adapt the licensed work, including for commercial use, provided that they cite the original article. This increases the value of articles published in *The R Journal*, whilst ensuring that authors receive appropriate credit for their work.

In the immediate term, the new licensing policy will mean that members of the community are free to translate articles in the journal, an initiative already planned by the R User's Group in South Korea (<http://www.openstatistics.net/>). It also opens up other possibilities, such as distributing sources alongside articles, that could be considered in future.

The selection of contributed articles in this issue is again testament to the diverse concerns of the R community. In the area of statistical methodology, Taylor Arnold and John Emerson introduce new functions for non-parametric goodness-of-fit tests, whilst Ian Marschner and joint authors Haizhou Wang and Mingzhou Song propose improved algorithms for generalized linear models and 1-D k means clustering respectively. On graphics, Han Lin Shang introduces the rainbow package for visualizing time series; Markus Gesmann and Diego de Castillo demonstrate use of the Google visualisation API with R, and Maxime Hervé presents a GUI designed to help R novices create publication-quality graphics in R. Providing easier access to R's functionality is also the focus of our first article, which demonstrates how to

create an Excel application from an R package, using `rpart` as an example. Finally the area of documentation and reproducible research is represented by articles on the spell-checking of Rd files and vignettes, and the implementation of the compendium concept with Sweave and DOCSTRIP.

In addition to these articles, we have the first Programmer's Niche article since becoming *The R Journal*. This column is intended to provide "programming pearls" for R developers—short articles that elucidate programming concepts or technical issues in developing R packages. In this instance, the article is not on R itself, but on writing portable C++ for R packages, problems with which cause many an error, especially on Solaris systems.

In the usual round up of news, readers are particularly encouraged to take a look at the report on R's Participation in the Google Summer of Code 2011. As well as reporting back on an especially productive season, the GSoC 2011 administrators discuss the costs and benefits of involvement in this scheme and note that suggestions for 2012 projects are already being collected.

Finally I would like to thank the associate editors and column editors for their work and support during my year's tenure as Editor-in-Chief. In 2012, this role will be taken on by Martyn Plummer. Peter Dalgaard will stand down from the editorial board after four years of service and Deepayan Sarkar will join us to fill his place. As Deepayan is currently based in Delhi, the editorial board will thus span three continents, with myself and Martyn in Europe and Hadley Wickham in the US. Such international diversity is important, when the R community itself is spread around the globe. Hopefully as the R Journal continues to establish itself and we look to expand the editorial board, the diversity of the R community will be increasingly represented.

*Heather Turner
Statistics Department,
University of Warwick,
Coventry, UK
Heather.Turner@R-project.org*

Creating and Deploying an Application with (R)Excel and R

Thomas Baier, Erich Neuwirth and Michele De Meo

Abstract We present some ways of using R in Excel and build an example application using the package **rpart**. Starting with simple interactive use of **rpart** in Excel, we eventually package the code into an Excel-based application, hiding all details (including R itself) from the end user. In the end, our application implements a service-oriented architecture (SOA) with a clean separation of presentation and computation layer.

Motivation

Building an application for end users is a very challenging goal. Building a statistics application normally involves three different roles: application developer, statistician, and user. Often, statisticians are programmers too, but are only (or mostly) familiar with statistical programming (languages) and definitely are not experts in creating rich user interfaces/applications for (casual) users.

For many—maybe even for most—applications of statistics, Microsoft Excel is used as the primary user interface. Users are familiar with performing simple computations using the spreadsheet and can easily format the result of the analyses for printing or inclusion in reports or presentations. Unfortunately, Excel does not provide support for doing more complex computations and analyses and also has documented weaknesses for certain numerical calculations.

Statisticians know a solution for this problem, and this solution is called R. R is a very powerful programming language for statistics with lots of methods from different statistical areas implemented in various packages by thousands of contributors. But unfortunately, R's user interface is not what everyday users of statistics in business expect.

The following sections will show a very simple approach allowing a statistician to develop an easy-to-use and maintainable end-user application. Our example will make use of R and the package **rpart** and the resulting application will completely hide the complexity and the R user interface from the user.

rpart implements recursive partitioning and regression trees. These methods have become powerful tools for analyzing complex data structures and have been employed in the most varied fields: CRM, financial risk management, insurance, pharmaceuticals and so on (for example, see: Altman (2002), Hastie et al. (2009), Zhang and Singer (1999)).

The main reason for the wide distribution of tree-based methods is their simplicity and intuitiveness.

Prediction of a quantitative or categorical variable, is done through a tree structure, which even non-professionals can read and understand easily. The application of a computationally complex algorithm thus results in an intuitive and easy to use tool. Prediction of a categorical variable is performed by a *classification tree*, while the term *regression tree* is used for the estimation of a quantitative variable.

Our application will be built for Microsoft Excel and will make use of R and **rpart** to implement the functionality. We have chosen Excel as the primary tool for performing the analysis because of various advantages:

- Excel has a familiar and easy-to-use user interface.
- Excel is already installed on most of the workstations in the industries we mentioned.
- In many cases, data collection has been performed using Excel, so using Excel for the analysis seems to be the logical choice.
- Excel provides many features to allow a high-quality presentation of the results. Pre-configured presentation options can easily be adapted even by the casual user.
- Output data (mostly graphical or tabular presentation of the results) can easily be used in further processing—e.g., embedded in PowerPoint slides or Word documents using OLE (a subset of COM, as in Microsoft Corporation and Digital Equipment Corporation (1995)).

We are using R for the following reasons:

- One cannot rely on Microsoft Excel's numerical and statistical functions (they do not even give the same results when run in different versions of Excel). See McCullough and Wilson (2002) for more information.
- We are re-using an already existing, tested and proven package for doing the statistics.
- Statistical programmers often use R for performing statistical analysis and implementing functions.

Our goal for creating an **RExcel**-based application is to enable any user to be able to perform the computations and use the results without any special knowledge of R (or even of **RExcel**). See Figure 1 for an example of the application's user interface. The results of running the application are shown in Figure 2.

The screenshot shows a Microsoft Excel spreadsheet with data in rows 1 through 12. Row 1 contains column headers V1 through V15. Rows 2 and 3 contain data points b and a respectively, with columns V2 through V15 filled with various values like '58.67', '0.5 u', 'g m bb', and '0 t f'. To the right of the spreadsheet is a prediction output box containing the text: "predict the class attribute (credit approval) to the credit card application". Below the output box is a button labeled "Clear output".

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
1	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15									
2	b	58.67	0.5 u	g m bb	0 t f		0 f	g	0	0														
3	a	58.67	0.5 u	g m bb	0 t f		0 f	g	0	112														
4																								
5																								
6																								
7																								
8																								
9																								
10																								
11																								
12																								

Figure 1: Basic user interface of the **RExcel** based end-user application.

The screenshot shows the same Microsoft Excel spreadsheet as Figure 1, but with additional data in the R, S, and T columns. The R column contains 'predGroup' and '+'. The S column contains '- 0.38 0.62' and '+ 0.38 0.62'. The T column contains '+'. To the right of the spreadsheet is a prediction output box containing the text: "predict the class attribute (credit approval) to the credit card application". Below the output box is a button labeled "Clear output".

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
1	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15		predGroup	-	+					
2	b	58.67	0.5 u	g m bb	0 t f		0 f	g	0	0							+	0.38	0.62					
3	a	58.67	0.5 u	g m bb	0 t f		0 f	g	0	112							+	0.38	0.62					
4																								
5																								
6																								
7																								
8																								
9																								
10																								
11																								
12																								

Figure 2: Computation results.

There are a few alternatives to **RExcel** for connecting Excel with R:

XLoop provides an Excel Add-In which is able to access various programming languages from within formulae. It supports languages like R, Javascript and others. The “backend” languages are accessed using TCP/IP communication. In contrast to this, **RExcel** uses COM, which has a very low latency (and hence is very fast). Additionally, the architecture of **RExcel** supports a completely invisible R server process (which is what we need for deploying the application), or can provide access to the R window while developing/testing.

inference for R provides similar functionality as **RExcel**. From the website, the latest supported R version is 2.9.1, while **RExcel** supports current R versions immediately after release (typically new versions work right out of the box without an update in **RExcel** or **statconnDCOM**).

Integrating R and Microsoft Excel

Work on integration of Excel and R is ongoing since 1998. Since 2001, a stable environment is available for building Excel documents based upon statistical methods implemented in R. This environment consists of plain R (for Windows) and a toolbox called **statconnDCOM** with its full integration with Excel using **RExcel** (which is implemented as an add-in for Microsoft Excel).

Like all Microsoft Office applications and some third-party applications, Excel provides a powerful

scripting engine combined with a highly productive visual development environment. This environment is called Visual Basic for Applications, or VBA for short (see Microsoft Corporation (2001)). VBA allows creation of “worksheet-functions” (also called user defined functions or UDFs), which can be used in a similar way as Excel’s built-in functions (like, e.g., “SUM” or “MEAN”) to dynamically compute values to be shown in spreadsheet cells, and Subs which allow arbitrary computations, putting results into spreadsheet ranges. VBA code can be bundled as an “Add-In” for Microsoft Excel which provides an extension both in the user interface (e.g., new menus or dialogs) and in functionality (additional sheet-functions similar to the built-in sheet functions).

Extensions written in VBA can access third-party components compatible with Microsoft’s Component Object Model (COM, see Microsoft Corporation and Digital Equipment Corporation (1995)). The link between R and Excel is built on top of two components: **RExcel** is an Add-In for Microsoft Excel implementing a spreadsheet-style interface to R and **statconnDCOM**. **statconnDCOM** exposes a COM component to any Windows application which encapsulates R’s functionality in an easy-to-use way.

statconnDCOM is built on an extensible design with exchangeable front-end and back-end parts. In the context of this article, the back-end is R. A back-end implementation for Scilab (INRIA (1989)) is also available. The front-end component is the COM interface implemented by **statconnDCOM**. This implementation is used to integrate R or Scilab into Windows applications. A first version of an Uno (OpenOffice.org (2009)) front-end has already been released for testing and download. Using this front-end R and

Scilab can easily be integrated into OpenOffice.org applications, like Calc (see OpenOffice.org (2006)). **ROOo** is available via Drexel (2009) and already supports Windows, Linux and MacOS X.

RExcel supports various user interaction modes:

Scratchpad and data transfer mode: Menus control data transfer from R to Excel and back; commands can be executed immediately, either from Excel cells or from R command line

Macro mode: Macros, invisible to the user, control data transfer and R command execution

Spreadsheet mode: Formulas in Excel cells control data transfer and command execution, automatic recalculation is controlled by Excel

Throughout the rest of this article, we will describe how to use *scratchpad mode* and *macro mode* for prototyping and implementing the application.

For more information on **RExcel** and **statconn-DCOM** see Baier and Neuwirth (2007).

Implementing the classification and regression tree

With **RExcel**'s tool set we can start developing our application in Excel immediately. **RExcel** has a "scratchpad" mode which allows you to write R code directly into a worksheet and to run it from there. Scratchpad mode is the user interaction mode we use when prototyping the application. We will then transform the prototypical code into a "real" application. In practice, the prototyping phase in scratchpad mode will be omitted for simple applications. In an Excel hosted R application we also want to transfer data between Excel and R, and transfer commands may be embedded in R code. An extremely simplistic example of code in an R code scratchpad range in Excel might look like this:

```
#!rput    inval      'Sheet1'!A1
result<-sin(inval)
#!rget    result      'Sheet1'!A2
```

R code is run simply by selecting the range containing code and choosing Run R Code from the pop-up menu.

Lines starting with `#!` are treated as special **RExcel** commands. `rput` will send the value (contents) of a cell or range to R, `rget` will read a value from R and store it into a cell or range.

In the example, the value stored in cell A1 of sheet Sheet1 will be stored in the R variable `inval`. After evaluating the R expression `result<-sin(inval)`, the value of the R variable `result` is stored in cell A2.

Table 1 lists all special **RExcel** commands and provides a short description. More information can be found in the documentation of **RExcel**.

In the example below, we have a dataset on the credit approval process for 690 subjects. For each record, we have 15 input variables (qualitative and quantitative) while variable 16 indicates the outcome of the credit application: positive (+) or negative (-). Based on this training dataset, we develop our classification tree which is used to show the influence of 15 variables on the loan application and to distinguish "good" applicants from "risky" applicants (so as to estimate variable 16). The risk manager will use this model fully integrated into Excel for further credit approval process.

The goal of the code snippet below is to estimate the model and display the chart with the classification tree. The risk manager can easily view the binary tree and use this chart to highlight the splits. This will help discover the most significant variables for the credit approval process. For example, in this case it is clear that the predictor variables `V9` determines the best binary partition in terms of minimizing the "impurity measure". In addition, the risk manager will notice that when `V9` equals `a`, the only significant variable to observe is `V4`. The graphical representation of the classification tree is shown in Figure 3 on page 8.

```
library(rpart)
#!rputdataframe    trainingdata          \
'database'!A1:P691
fit<-rpart(V16~.,data=trainingdata)
plot(fit,branch=0.1,uniform=T,margin=.1, \
compress=T,nspace=0.1)
text(fit,fancy=T,use.n=T)
#!insertcurrentplot    'database'!T10
graphics.off()
```

Note: \ in the code denotes a line break for readability and should not be used in the real spreadsheet

After estimating the model and visualizing the results, you can use the Classification Tree to make predictions about the variable V16: the applicants will be classified as "good" or "risky." By running the following code, the 15 observed variables for two people are used to estimate the probability of credit approval. Generally, a "positive" value (+) with probability higher than 0.5 will indicate to grant the loan. So the risk manager using this model will decide to grant the credit in both cases.

```
library(rpart)
#!rputdataframe trainingdata          \
'database'!A1:P691
#!rputdataframe newcases            \
'predict'!A1:O3
outdf<-as.data.frame(predict(fit,newcases))
predGroup <- ifelse(outdf[,1]>0.5, \
names(outdf[1]),names(outdf[2]))
res<-cbind(predGroup,outdf)
#!rgetdataframe res      'predict'!R1
```

<i>Command</i>	<i>Description</i>
<code>#!rput variable range</code>	store the value (contents) of a range in an R variable
<code>#!rputdataframe variable range</code>	store the value of a range in an R data frame
<code>#!rputpivotable variable range</code>	store the value of a range in an R variable
<code>#!rget r-expression range</code>	store the value of the R expression in the range
<code>#!rgetdataframe r-expression range</code>	store the data frame value of the R expression in the range
<code>#!insertcurrentplot cell-address</code>	insert the active plot into the worksheet

Table 1: Special RExcel commands used in sheets

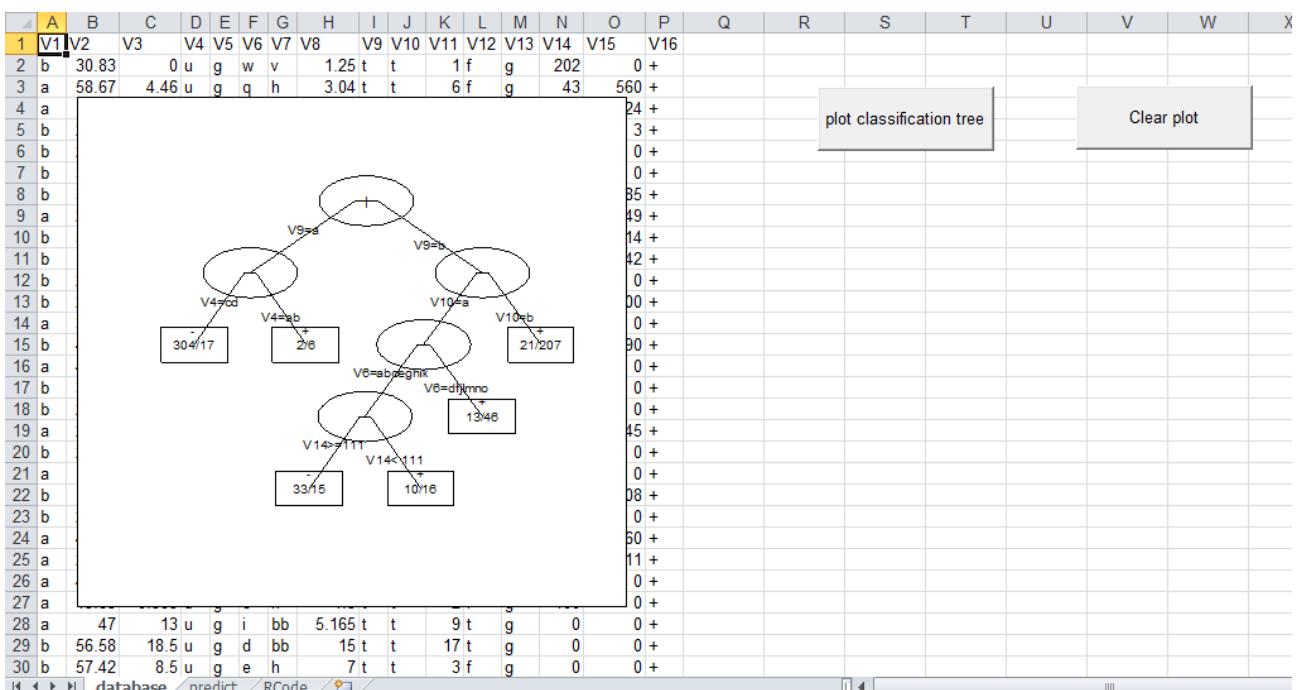


Figure 3: Graphical representation of the classification tree in the RExcel based application.

The potential of this approach is obvious: Very complex models, such as classification and regression trees, are made available to decision makers (in this case, the risk manager) directly in Microsoft Excel. See Figure 2 on page 6 for a screen-shot of the presentation of the results.

Tidying up the spreadsheet

In an application to be deployed to end users, R code should not be visible to the users. Excel's mechanism for performing operations on data is to run macros written in VBA (Visual Basic for Applications, the programming language embedded in Excel). **RExcel** implements a few functions and subroutines which can be used from VBA. Here is a minimalistic example:

```
Sub RExcelDemo()
    RInterface.StartRServer
    RInterface.GetRAppl "sin", _
        Range("A2"), Range("A1")
    RInterface.StopRServer
End Sub
```

`GetRAppl` applies an R function to arguments taken from Excel ranges and puts the result in a range. These arguments can be scalars, vectors, matrices or dataframes. In the above example, the function `sin` is applied to the value from cell A1 and the result is put in cell A2. The R function given as the first argument to `GetRAppl` does not necessarily have to be a named function, any function expression can be used.

RExcel has several other functions that may be called in VBA macros. `RRun` runs any code given as string. `RPut` and `RGet` transfer matrices, and `RPutDataframe` and `RGetDataframe` transfer dataframes. `RunRCall` similar to `GetRAppl` calls an R function but does not transfer any return value to Excel. A typical use of `RunRCall` is for calling plot functions in R. `InsertCurrentRPlot` embeds the current R plot into Excel as an image embedded in a worksheet.

In many cases, we need to define one or more R functions to be used with `GetRAppl` or the other VBA support functions and subroutines. **RExcel** has a mechanism for that. When **RExcel** connects to R (using the command `RInterface.StartRServer`), it will check whether the directory containing the active workbook also contains a file named `RExcelStart.R`. If it finds such a file, it will read its contents and evaluate them with R (source command in R). After doing this, **RExcel** will check if the active workbook contains a worksheet named `RCode`. If such a worksheet exists, its contents also will be read and evaluated using R.

Some notes on handling R errors in Excel: The R implementation should check for all errors which are expected to occur in the application. The application itself is required to pass correctly typed arguments

when invoking an R/ function. If **RExcel** calls a function in R and R throws an error, an Excel dialog box will pop up informing the user of the error. An alternative to extensive checking is to use R's `try` mechanism to catch errors in R and handle them appropriately.

Using these tools, we now can define a macro performing the actions our scratchpad code did in the previous section. Since we can define auxiliary functions easily, we can also now make our design more modular.

The workhorse of our application is the following VBA macro:

```
Sub PredictApp()
    Dim outRange As Range
    ActiveWorkbook.Worksheets("predict") _
        .Activate
    If Err.Number <> 0 Then
        MsgBox "This workbook does not " _
            & "contain data for rpartDemo"
        Exit Sub
    End If
    ClearOutput
    RInterface.StartRServer
    RInterface.RunRCodeFromRange
    ThisWorkbook.Worksheets("RCode") _
        .UsedRange

    RInterface.GetRAppl
    "function(" _
        & "trainingdata,groupvarname," _
        & "newdata)predictResult(fitApp("_
        & "trainingdata,groupvarname)," _
        & "newdata)", _
    ActiveWorkbook.Worksheets( _
        "predict").Range("R1"),
    AsSimpleDF(DownRightFrom( _
        ThisWorkbook.Worksheets( _
            "database").Range("A1"))),
    "V16",
    AsSimpleDF(ActiveWorkbook
        .Worksheets("predict").Range(
        "A1").CurrentRegion)

    RInterface.StopRServer
    Set outRange = ActiveWorkbook
        .Worksheets("predict").Range("R1") _
        .CurrentRegion
    HighLight outRange
End Sub
```

The function `predictResult` is defined in the worksheet `RCode`.

Packaging the application

So far, both our code (the R commands) and data have been part of the same spreadsheet. This may be convenient while developing the **RExcel**-based application

or if you are only using the application for yourself, but it has to be changed for redistribution to a wider audience.

We will show how to divide the application into two parts, the first being the Excel part, the second being the R part. With a clear interface between these, it will be easy to update one of them without affecting the other. This will make testing and updating easier. Separating the R implementation from the Excel (or **RExcel**) implementation will also allow an R expert to work on the R part and an Excel expert to work on the Excel part.

As an additional benefit, exchanging the Excel front-end with a custom application (e.g., written in a programming language like C#) will be easier, too, as the R implementation will not have to be changed (or tested) in this case.

The application's R code interface is simple, yet powerful. **RExcel** will only have to call a single R function called `approval`. Everything else is hidden from the user (including the training data). `approval` takes a data frame with the data for the cases to be decided upon as input and returns a data frame containing the group classification and the probabilities for all possible groups. Of course, the return value can also be shown in a figure.

Creating an R package

The macro-based application built in section “[Tidying up the spreadsheet](#)” still contains the R code and the training data. We will now separate the implementation of the methodology and the user interface by putting all our R functions and the training data into an R package. This R package will only expose a single function which gets data for the new cases as input and returns the predicted group membership as result. Using this approach, our application now has a clear architecture. The end user workbook contains the following macro:

```
Sub PredictApp()
    Dim outRange As Range
    ClearOutput
    RInterface.StartRServer

    RInterface.RRun "library(RExcelrpart)"
    RInterface.GetRAppl "approval", _
        Range("'predict'!R1"), _
        AsSimpleDF(Range("predict!A1") _
            .CurrentRegion)

    RInterface.StopRServer
    Set outRange = Range("predict!R1") _
        .CurrentRegion
    HighLight outRange
End Sub
```

In this macro, we start a connection to R, load the R package and call the function provided by this pack-

age and then immediately close the the connection to R.

The **statconnDCOM** server can reside on another machine than the one where the Excel application is running. The server to be used by **RExcel** can be configured in a configuration dialog or even from within VBA macros. So with minimal changes, the application created can be turned into an application which uses a remote server. A further advantage of this approach is that the R functions and data used by the application can be managed centrally on one server. If any changes or updates are necessary, the Excel workbooks installed on the end users' machines do not need to be changed.

Building a VBA add-in for Excel

The implementation using an R package still has some shortcomings. The end user Excel workbook contains macros, and often IT security policies do to not allow end user workbooks to contain any executable code (macros). Choosing a slightly different approach, we can put all the VBA macro code in an Excel add-in. In the end user workbook, we just place buttons which trigger the macros from the add-in. When opening a new spreadsheet to be used with this add-in, Excel's template mechanism can be used to create the buttons on the new worksheet. The code from the workbook described in section “[Tidying up the spreadsheet](#)” cannot be used “as is” since it is written under the assumption that both the training data and the data for the new cases are contained in the same workbook. The necessary changes, however, are minimal. Converting the workbook from section “[Tidying up the spreadsheet](#)” again poses the problem that the data and the methodology are now deployed on the end users' machines. Therefore updating implies replacing the add-in on all these machines. Combining the add-in approach with the packaging approach from section “[Creating an R package](#)” increases modularization. With this approach we have:

- End user workbooks without any macro code.
- Methodology and base data residing on a centrally maintained server.
- Connection technology for end users installed for all users in one place, not separately for each user.

Deploying the Application

Using the application on a computer requires installation and configuration of various components.

The required (major) components are:

- Microsoft Excel, including **RExcel** and **statconnDCOM**
- R, including **rscproxy**

- The VBA Add-In and the R package created throughout this article

For the simplest setup, all components are installed locally. As an alternative, you can also install Excel, **RExcel** and our newly built VBA Add-In on every workstation locally and install everything else on a (centralized) server machine. In this case, R, **rscproxy**, our application's R package and **statconnDCOM** are installed on the server machine and one has to configure **RExcel** to use R via **statconnDCOM** on a remote server machine. Please beware that this kind of setup can be a bit tricky, as it requires a correct DCOM security setup (using the Windows tool *dcomcnfg*).

Downloading

All examples are available for download from the *Download* page on <http://rcom.univie.ac.at>.

Bibliography

- E. I. Altman. *Bankruptcy, Credit Risk, and High Yield Junk Bonds*. Blackwell Publishers Inc., 2002.
- T. Baier. *rcom: R COM Client Interface and internal COM Server*, 2007. R package version 1.5-1.
- T. Baier and E. Neuwirth. *R (D)COM Server V2.00*, 2005. URL <http://cran.r-project.org/other/DCOM>.
- T. Baier and E. Neuwirth. Excel :: COM :: R. *Computational Statistics*, 22(1):91–108, April 2007. URL <http://www.springerlink.com/content/uv6667814108258m/>.
- Basel Committee on Banking Supervision. International Convergence of Capital Measurement and Capital Standards. Technical report, Bank for International Settlements, June 2006. URL <http://www.bis.org/publ/bcbs128.pdf>.
- L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.
- J. M. Chambers. *Programming with Data*. Springer, New York, 1998. URL <http://cm.bell-labs.com/cm/ms/departments/sia/Sbook/>. ISBN 0-387-98503-4.
- R. Drexel. *ROOo*, 2009. URL <http://rcom.univie.ac.at/>.
- T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009.
- INRIA. *Scilab*. INRIA ENPC, 1989. URL <http://www.scilab.org>.
- B. D. McCullough and B. Wilson. On the accuracy of statistical procedures in Microsoft Excel 2000 and Excel XP. *Computational Statistics and Data Analysis*, 40:713–721, 2002.
- Microsoft Corporation. Microsoft Office 2000/Visual Basic Programmer's Guide. In *MSDN Library*, volume Office 2000 Documentation. Microsoft Corporation, October 2001. URL <http://msdn.microsoft.com/>.
- Microsoft Corporation and Digital Equipment Corporation. The component object model specification. Technical Report 0.9, Microsoft Corporation, October 1995. Draft.
- OpenOffice.org. OpenOffice, 2006. URL <http://www.openoffice.org/>.
- OpenOffice.org. Uno, 2009. URL <http://wiki.services.openoffice.org/wiki/Uno/>.
- H. Zhang and B. Singer. *Recursive Partitioning in the Health Sciences*. Springer-Verlag, New York, 1999. ISBN 0-387-98671-5.

Thomas Baier
Department of Scientific Computing
University of Vienna
1010 Vienna
Austria
thomas.baier@univie.ac.at

Erich Neuwirth
Department of Scientific Computing
University of Vienna
1010 Vienna
Austria
erich.neuwirth@univie.ac.at

Michele De Meo
Venere Net Spa
Via della Camilluccia, 693
00135 Roma
Italy
micheledemeo@gmail.com

glm2: Fitting Generalized Linear Models with Convergence Problems

by Ian C. Marschner

Abstract The R function `glm` uses step-halving to deal with certain types of convergence problems when using iteratively reweighted least squares to fit a generalized linear model. This works well in some circumstances but non-convergence remains a possibility, particularly with a non-standard link function. In some cases this is because step-halving is never invoked, despite a lack of convergence. In other cases step-halving is invoked but is unable to induce convergence. One remedy is to impose a stricter form of step-halving than is currently available in `glm`, so that the deviance is forced to decrease in every iteration. This has been implemented in the `glm2` function available in the `glm2` package. Aside from a modified computational algorithm, `glm2` operates in exactly the same way as `glm` and provides improved convergence properties. These improvements are illustrated here with an identity link Poisson model, but are also relevant in other contexts.

It is not too uncommon for iteratively reweighted least squares (IRLS) to exhibit convergence problems when fitting a generalized linear model (GLM). Such problems tend to be most common when using a non-standard link function, such as a log link binomial model or an identity link Poisson model. Consequently, most commonly used statistical software has the provision to invoke various modifications of IRLS if non-convergence occurs.

In the `stats` package of R, IRLS is implemented in the `glm` function via its workhorse routine `glm.fit`. This routine deals with specific types of convergence problems by switching to step-halving if iterates display certain undesirable properties. That is, if a full Fisher scoring step of IRLS will lead to either an infinite deviance or predicted values that are invalid for the model being fitted, then the increment in parameter estimates is repeatedly halved until the updated estimates no longer exhibit these features. This is achieved through repeated application of the call

```
start <- (start + coefold)/2
```

where `coefold` and `start` contain estimates from the previous and current iterations, respectively.

Although this approach works well in some contexts, it can be prone to fail in others. In particular, although the step-halving process in `glm.fit` will throw an errant iterative sequence back into the desired region, the sequence may repeatedly try to escape that region and never converge. Furthermore, it is even

possible for the IRLS iterative sequence to be such that step-halving is never invoked in `glm.fit`, yet the sequence does not converge. Such behavior is typically accompanied by a deviance sequence that increases in one or more of the iterations. This suggests a modification to `glm.fit` which has been implemented in the `glm2` package (Marschner, 2011).

As motivation for the proposed modification, we begin by discussing the potential for non-convergence using some numerical examples of the above types of behavior. The `glm2` package is then discussed, which consists of a main function `glm2` and a workhorse routine `glm.fit2`. These are modified versions of `glm` and `glm.fit`, in which step-halving is used to force the deviance to decrease from one iteration to the next. It is shown that this modification provides improved convergence behavior.

Non-convergence

We start by providing illustrations of the two types of non-convergence alluded to above. Specifically, we will consider situations in which standard IRLS does not converge, and for which either: (i) the step-halving in `glm.fit` is invoked but cannot induce convergence; or (ii) the step-halving in `glm.fit` is never invoked despite the non-convergence of IRLS. These two types of behavior will be illustrated using an identity link Poisson regression model, which can be prone to convergence problems as the link function does not automatically respect the non-negativity of the Poisson means. This context is useful for studying the convergence properties of algorithms based on IRLS, because for this particular GLM a reliable alternative algorithm exists which is not based on IRLS, as described by Marschner (2010). While we focus on the identity link Poisson model here, the same behavior can occur in other models and the documentation for the `glm2` package includes a log link binomial example.

In Table 3.2 of Agresti (2007) a data set is presented that is amenable to analysis using a linear Poisson model. The data set consists of a count response variable y_i , $i = 1, \dots, 173$, called Sa in Agresti (2007), with observed values ranging from 0 through 15 and a mean of 2.9. Also presented are a number of potential covariates. Here we define three covariates, x_{i1} , x_{i2} and x_{i3} , $i = 1, \dots, 173$, similarly to Marschner (2010). The first two of these covariates, x_{i1} and x_{i2} , are defined by dichotomizing the covariates called C and S in Agresti (2007), such that $\{x_{i1}\} = 1\{C > 3\}$ and $\{x_{i2}\} = 1\{S < 3\}$. This yields two 0/1 binary covariates with means 0.4 and 0.3, respectively. The third

covariate, x_{i3} , is a continuous covariate called W in Agresti (2007), which is shifted here by subtracting the smallest value, so that it ranges from 0 through 12.5 with a mean of 5.3.

Assuming y_i is an observation from a Poisson distribution with mean μ_i , the identity link model is

$$\mu_i = \alpha_0 + \alpha_1 x_{i1} + \alpha_2 x_{i2} + \alpha_3 x_{i3}$$

with parameter vector $\theta = (\alpha_0, \alpha_1, \alpha_2, \alpha_3)$. This model can be fitted in R by first defining a response vector y with $y[i] = y_i$, and corresponding covariate vectors x_1 , x_2 and x_3 with $c(x1[i], x2[i], x3[i]) = (x_{i1}, x_{i2}, x_{i3})$, along with a data frame `poisdata` through the assignment `poisdata <- data.frame(y, x1, x2, x3)`. The call to fit the model is then

```
glm(y ~ x1 + x2 + x3, data = poisdata,
  family = poisson(link = "identity"),
  start = rep(1, 4))
```

which includes an initial estimate through the `start` argument because the default choice is invalid for this model. Below we also use the `control` argument to monitor the iterative behavior.

As discussed later in the paper, the above call produces satisfactory convergence for these data; however, a bootstrap analysis using the same call leads to quite severe numerical instability. Using the `sample` function, bootstrap replications were generated by sampling with replacement from the original data, after which `glm` was applied to each replication. This process readily produced replications for which `glm` failed, and a collection of 100 such replications was generated in which non-convergence occurred. Two of these replications are discussed as examples in this section, while the full collection of 100 replications is discussed further in the next section.

Figure 1 displays the lack of convergence using `glm` for the two illustrative replications. Increasing the maximum number of iterations does not alleviate the problem. Also plotted in Figure 1 is the deviance achieved by the maximum likelihood estimate (MLE), calculated using the non-IRLS linear Poisson method of Marschner (2010). For Figure 1(a) this minimum possible deviance is 604.0 with an MLE of $\hat{\theta} = (-0.095, -0.385, 0.618, 0.530)$, while for Figure 1(b) the minimum possible deviance is 656.3 with an MLE of $\hat{\theta} = (0.997, -1.344, -0.169, 0.524)$. Inspection of the score functions reveals both MLEs are stationary and in the interior of the parameter space.

These two examples illustrate the two scenarios of non-convergence described at the beginning of this section. In Figure 1(a), step-halving was invoked in 28 of the 100 iterations, showing that `glm` can fail to converge even with step-halving. In Figure 1(b) step-halving was not invoked, showing that `glm` can fail to converge without ever making use of step-halving. The latter example is indicative of a potential problem with Newton-type algorithms, which can have a

so-called attracting periodic cycle. In this case IRLS is attracted to a cycle of two iterate values, with deviances of 673.2 and 691.1, and then subsequently oscillates between those values.

Although these results use a specific starting value, the non-convergence cannot be remedied with better initial estimates. This is illustrated in Figure 1(a), where the iterates get very close to the optimal value. Only when the initial estimate is identical to the MLE does `glm` converge (in one iteration).

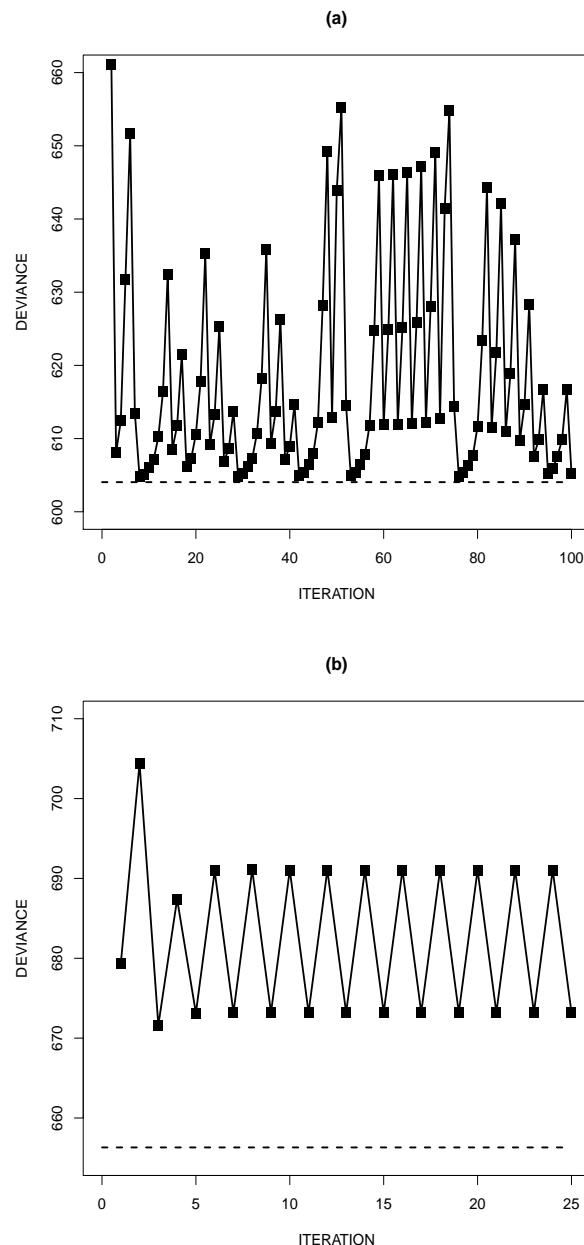


Figure 1: Examples of non-convergence in `glm`. Dashed lines denote the optimal deviance.

The `glm2` package

An important feature of Figure 1 is that the iterative sequences display instances of increasing deviance

from one iteration to the next. It is well known that step-halving can be used in Newton-type algorithms to force the objective function to improve monotonically (Lange, 2010, p. 251). This approach is used to improve the convergence properties of IRLS in other standard statistical software. Here we discuss its implementation in the `glm2` function available within the `glm2` package, and the associated improvements in convergence properties.

The source code for `glm.fit` has two step-halving blocks called inner loops 1 and 2. These occur within the main loop immediately prior to the test of convergence, and they use step-halving to rectify a divergent deviance and invalid predicted values, respectively. The main change implemented in the `glm2` package is that the modified routine `glm.fit2` has a further step-halving block, called inner loop 3, which tests whether the deviance is lower than in the previous iteration. If not, step-halving is invoked until the deviance is lowered, leading to an iterative sequence that monotonically decreases the deviance.

Convergence in `glm.fit` occurs if `abs(rel) < control$epsilon` where `rel` is the relative change `(dev - devold) / (0.1 + abs(dev))` from the current deviance `devold` to the updated deviance `dev`, and `control$epsilon` is a positive tolerance. In `glm.fit2` the same convergence test is used, and if it is not satisfied because `rel <= -control$epsilon`, then iterations proceed as in `glm.fit`. However, if convergence is not satisfied because `rel >= control$epsilon`, then inner loop 3 invokes step-halving until `rel <= -control$epsilon`. Thus, as well as decreasing the deviance, the step-halving takes `rel` from one side of the convergence region to the other, therefore never causing false convergence.

The `glm2` function is essentially identical to `glm`, except the default fitting method is `glm.fit2`. This allows `glm2` to be called with the same calling sequence as `glm`. Alternatively, in version 2.12.1 and later, it should be possible to achieve the same effect by passing `glm.fit2` to `glm` through the `method` argument, instead of the default method `glm.fit`. Indeed, existing scripts calling `glm` should work unchanged with the new fitting function, after first executing

```
glm <- function(..., method = glm2::glm.fit2){
  stats::glm(..., method = method)
}
```

which makes `glm.fit2` the default fitting method when `glm` is called.

In the previous section we discussed a data set for which 100 bootstrap replications were generated where `glm` failed to converge. When `glm2` was used, convergence was achieved for all 100 replications with an initial value of 1 for each of the four parameters. This included the two illustrative examples plotted in Figure 1, which both converged to the MLEs produced by the non-IRLS method of Marschner (2010).

This convergence is displayed in Figure 2, together with the path that `glm` would take.

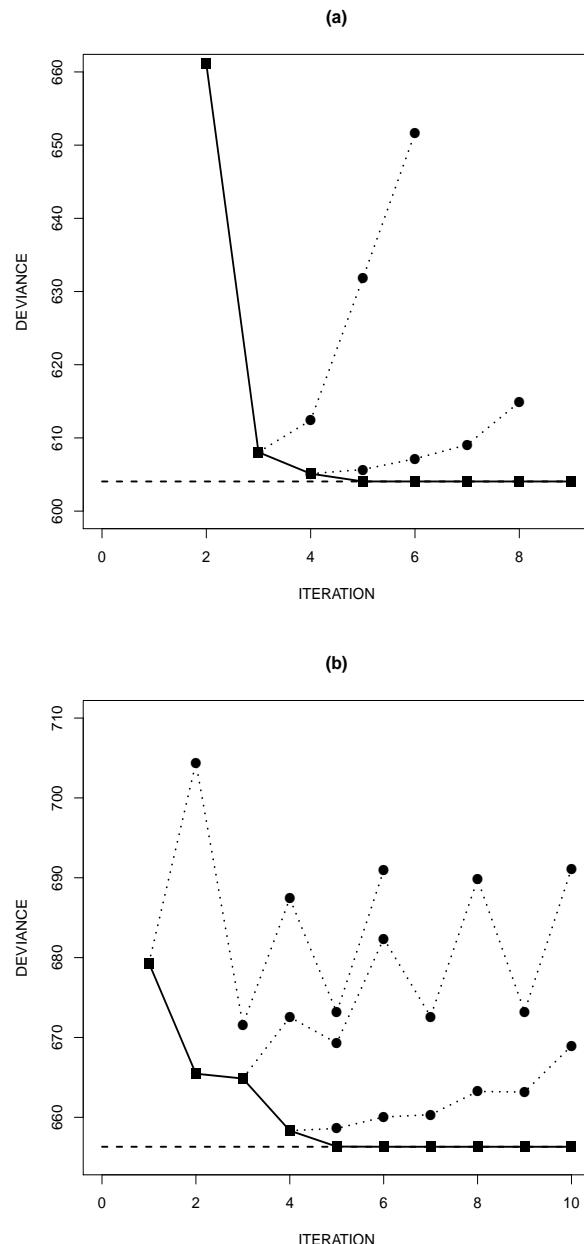


Figure 2: Convergence of `glm2` (squares) for the examples presented in Figure 1. Circles denote the path that `glm` would take.

Using `glm2`, Figure 3 provides medians and interquartile ranges of the estimates from the 100 replications that did not converge in `glm`. Also shown is the same information for 100 replications where `glm` did converge. In each case the two sets of estimates are relatively consistent. This would seem to suggest that estimates from the replications in which `glm` failed to converge are not particularly unusual compared to those for which it did converge. In particular, as demonstrated in Figure 1, non-convergence of `glm` can occur even when the MLE is a stationary point in the interior of the parameter space.

As well as the functions `glm2` and `glm.fit2`, the `glm2` package provides data sets and example code illustrating the usefulness of the proposed change. This allows reproduction of the behavior described here for the identity link Poisson model, and also provides an example for the log link binomial model.

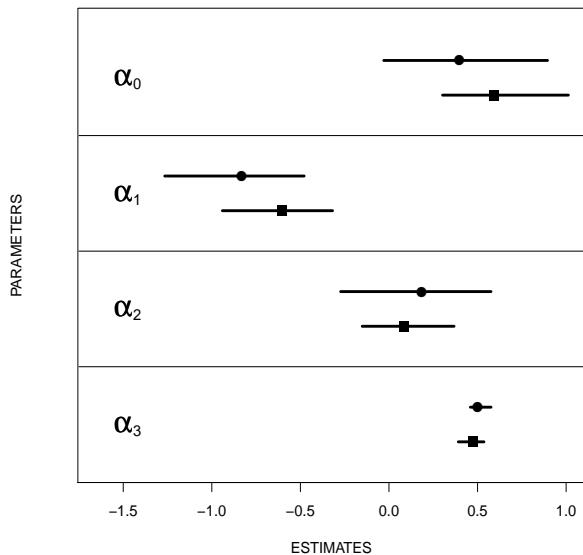


Figure 3: Medians and interquartile ranges of estimates in 100 replications that converged (squares) and 100 replications that did not converge (circles) in `glm`. The latter were calculated using `glm2`.

Discussion

This paper describes a modification that improves the convergence properties of `glm`, and which is available in the `glm2` function in the `glm2` package. Following the recommendation of the editors, the modification has been presented as a separate package rather than as a code snippet. This has the advantage of facilitating thorough testing of the proposed change, and also allows some example analyses to be made available through the package. These examples are promising, but additional testing would be required before considering a change to `glm` and its ancillaries in the standard R `stats` package.

While `glm2` should converge whenever `glm` converges, it may take a different path to convergence. This will occur whenever a convergent `glm` sequence has an iteration where the deviance increased. The fact that `glm2` may take a different path to convergence should be inconsequential in practice, particularly as it will be a more stable monotone path.

Although we have found cases in which `glm` has convergence problems, in other contexts it copes very well with numerical difficulties. The data set from which the bootstrap replications were generated is an example. Using an identity link Poisson model, the existing step-halving in `glm` allows it to converge to the

non-stationary MLE $\hat{\theta} = (0.578, -0.626, 0.048, 0.484)$. Since the fitted value for the observed covariate pattern $(x_{i1}, x_{i2}, x_{i3}) = (1, 1, 0)$ is $0.578 - 0.626 + 0.048 = 0$, and all other fitted values are positive, the estimate produced is on the boundary of the parameter space. Not all GLM software would be able to converge to a non-stationary boundary point such as this.

Finally, we end with an observation on predicted values when using a non-standard link function. When `glm` (or `glm2`) converges with a link function that does not respect the natural parameter restrictions of the error distribution, such as in the example here, the predicted values will meet these restrictions for all observed covariate combinations. However, predicted values for other covariate combinations may not meet these restrictions. This is true not only for extreme covariate values outside the observed ranges, but also for unobserved combinations that are within the observed ranges. For example, the values $x_{i1} = 1$, $x_{i2} = 0$ and $x_{i3} = 0$ are all observed in the above data set, but the combination $(x_{i1}, x_{i2}, x_{i3}) = (1, 0, 0)$ was not observed, and has a negative predicted value of $0.578 - 0.626 = -0.048$. Although not available in `glm` or `glm2`, in principle it is possible to use a smaller parameter space that only has valid predicted values for covariate combinations that are within the observed ranges of the individual covariates. For the above data set this leads to the estimate $\hat{\theta} = (0.640, -0.640, 0.000, 0.476)$ (Marschner, 2010), which is slightly different to the one produced by `glm` and has a valid predicted value for the combination $(1, 0, 0)$. A discussion of the relative merits of these parameter spaces is not attempted here, but it is important to understand which one is being used when one fits a GLM with a non-standard link.

Bibliography

- A. Agresti. *An Introduction to Categorical Data Analysis*. Wiley, Hoboken, USA, second edition, 2007.
- K. Lange. *Numerical Analysis for Statisticians*. Springer, New York, USA, second edition, 2010.
- I. C. Marschner. Stable computation of maximum likelihood estimates in identity link Poisson regression. *Journal of Computational and Graphical Statistics*, 19(3):666–683, 2010.
- I. C. Marschner. `glm2`: Fitting generalized linear models. 2011. URL <http://CRAN.R-project.org/package=glm2>. R package version 1.0.

*Ian Marschner
Department of Statistics
Macquarie University
NSW 2109
Australia
ian.marschner@mq.edu.au*

Implementing the Compendium Concept with Sweave and DOCSTRIP

by Michael Lundholm

Abstract This article suggests an implementation of the compendium concept by combining *Sweave* and the *L^AT_EX* literate programming environment *DOCSTRIP*.

Introduction

Gentleman and Lang (2007) introduced *compendiums* as a mechanism to combine text, data, and auxiliary software into a distributable and executable unit, in order to achieve *reproducible research*¹:

“...research papers with accompanying software tools that allow the reader to directly reproduce the result and employ the methods that are presented ...” Gentleman and Lang (2007, (abstract))

Gentleman (2005) provides an example of how the compendium concept can be implemented. The core of the implementation is a *Sweave*² source file. This source file is then packaged together with data and auxiliary software as an R package.

In this article I suggest an alternative implementation of the compendium concept combining *Sweave* with *DOCSTRIP*. The latter is the *L^AT_EX* literate programming environment used in package documentation and as an installation tool.³ *DOCSTRIP* has previously been mentioned in the context of reproducible research, but then mainly as a hard to use alternative to *Sweave*.⁴ Here, instead, *DOCSTRIP* and *Sweave* are combined.

Apart from the possibility to enjoy the functionality of *Sweave* and packages such as *xtable* etc the main additional advantages are that

- in many applications almost all code and data can be kept in a single source file,
- multiple documents (i.e., PDF files) can share the same *Sweave* code chunks.

This means not only that administration of an empirical project is facilitated but also that it becomes easier to achieve reproducible research. Since *DOCSTRIP* is a part of every *L^AT_EX* installation a *Sweave* user need not install any additional software. Finally, *Sweave* and *DOCSTRIP* can be combined to produce more complex projects such as R packages.

One example of the suggested implementation will be given. It contains R code common to more than one document; an article containing the advertisement of the research (using the terminology of Buckheit and Donoho (1995)), and one technical documentation of the same research. In the following I assume that the details of *Sweave* are known to the readers of the R Journal. The rest of the article will (i) give a brief introduction to *DOCSTRIP*, (ii) present and comment the example and (iii) close with some final remarks.

DOCSTRIP

Suppose we have a source file the entire or partial content of which should be tangled into one or more result files. In order to determine which part of the source file that should be tangled into a certain result file (i) the content of the source file is tagged with none, one or more tags (tag-lists) and (ii) the various tag-lists are associated with the result files in a *DOCSTRIP* “installation” file.

There are several ways to tag parts of the source file:

- A single line: Start the line with '%<tag-list>'.
- Several lines, for instance one or more code or text chunks in *Sweave* terminology: On a single line before the first line of the chunk enter the start tag '%<*tag-list>' and on a single line after the last line of the chunk the end tag '%</tag-list>'.
- All lines: Lines that should be in all result files are left untagged.

‘tag-list’ is a list of tags combined with the Boolean operators ‘|’ (logical or), ‘&’ (logical and) and ‘!’ (logical negation). A frequent type of list would be, say, ‘tag1|tag2|tag3’ which will tangle the tagged material whenever ‘tag1’, ‘tag2’ or ‘tag3’ is called for into the result files these tags are associated with. The initial ‘%’ of the tags must be in the file’s first column or else the tag will not be recognised as a *DOCSTRIP* tag. Also, tags must be matched so a start tag with ‘tag-list’ must be closed by an end tag with ‘tag-list’. This resembles the syntax of *L^AT_EX* environments rather than the *Sweave* syntax, where the end of a code or text chunk is indicated by the beginning of the next text or code chunk. Note also that tags cannot be nested.⁵

¹Buckheit and Donoho (1995).

²Leisch. See also Leisch (2008) and Meredith and Racine (2008).

³Mittelbach et al. (2005) and Goossens et al. (1994, section 14.2).

⁴Hothorn (2006) and Rising (2008).

⁵More exactly: Outer tags, which are described here, cannot be nested but inner tags can be nested with outer tags. See Goossens et al. (1994, p. 820) for details.

The following source file ('docex.txt') exemplifies all three types of tags:

```

1 %<file1|file2>This line begins both files.
2 %<*file1>
3
4 This is the text that should be included in file1
5
6 %</file1>
7
8 This is the text to be included in both files
9
10 %<*file2>
11 This is the text that should be included in file2
12 %</file2>
13 %<*file1|file2>
14 Also text for both files.
15 %</file1|file2>
```

For instance, line 1 is a single line tagged 'file1' or 'file2', line 2 starts and line 6 ends a tag 'file1' and line 13 starts and line 15 ends a tag 'file1' or 'file2'. Lines 7 – 9 are untagged.

The next step is to construct a DOCSTRIP installation file which associates each tag with one or more result files:

```

1 \input docstrip.tex
2 \keepsilent
3 \askforoverwritefalse
4 \nopreamble
5 \nopostamble
6 \generate{
7   \file{file1.txt}{\from{docex.txt}{file1}}
8   \file{file2.txt}{\from{docex.txt}{file2}}
9 }
10 \endbatchfile
```

Line 1 loads DOCSTRIP. Lines 2 – 5 contain options that basically tell DOCSTRIP not to issue any messages, to write over any existing result files and not to mess up the result files with pre- and post-ambls.⁶ The action takes place on lines 6 – 9 within the command '\generate{}', where lines 7 – 8 associate the tags 'file1' and 'file2' in the source file 'docex.txt' with the result files 'file1.txt' and 'file2.txt'.

We name this file 'docex.ins', where '.ins' is the conventional extension for DOCSTRIP installation files. DOCSTRIP is then invoked with

```
latex docex.ins
```

A log-file called 'docex.log' is created from which we here show the most important parts (lines 56 – 67):

```

56 Generating file(s) ./file1.txt ./file2.txt
57 \openout0 = `./file1.txt'.
58
```

⁶Pre- and postambls are text lines that are starting with a comment character. Since result files may be processed by software using different comment characters some care is needed to use pre- and postambls constructed by DOCSTRIP. See Goossens et al. (1994, p. 829f and 833f) how to set up pre- and postambls that are common to all result files from a given installation file.

⁷From the example one infer that multiple source files are possible, although the compendium implementation discussed later in most cases would have only one.

```

59 \openout1 = `./file2.txt'.
60
61
62 Processing file docex.txt (file1) -> file1.txt
63                                         (file2) -> file2.txt
64 Lines processed: 15
65 Comments removed: 0
66 Comments passed: 0
67 Codelines passed: 8
```

We see that two result files are created from the 15 lines of code in the source file. First 'file1.txt';

```

1 This line begins both files.
2
3 This is the text that should be included in file1
4
5
6 This is the text to be included in both files
7
8 Also text for both files.
```

and 'file2.txt';

```

1 This line begins both files.
2
3 This is the text to be included in both files
4
5 This is the text that should be included in file2
6 Also text for both files.
```

Note that some lines are blank in both the original source file and the result files. Disregarding these the two result files together have 8 lines of code. The untagged material in lines 7 – 9 in the source files is tangled into both result files, the blank lines 7 and 8 in the source file result in the blank lines 5 and 7 in 'file1.txt' and the blank lines 2 and 4 in 'file2.txt'.

Example

In the following a simple example will be given of how DOCSTRIP can be combined with Sweave to implement the compendium concept. The starting point is a "research problem" which involves loading some data into R, preprocessing the data, conducting an estimation and presenting the result. The purpose is to construct a single compendium source file which contains the code used to create (i) an "article" PDF-file which will provide a brief account of the test and (ii) a "technical documentation" PDF-file which gives a more detailed description of loading and preprocessing data and the estimation. The source file also contains the code of a BibTeX database file and the

DOCSTRIP installation file. Although this source file is neither a \LaTeX file or a *Sweave* file I will use the extension ‘.rnw’ since it first run through *Sweave*. Here we simplify the example by using data from an R package, but if the data set is not too large it could be a part of the source file.

We can think of the “article” as the “advertisement” intended for journal publication and the “technical documentation” as a more complete account of the actual research intended to be available on (say) a web place. However, tables, graphs and individual statistics should originate from the same R code so whenever *Sweave* is executed these are updated in both documents. There may also be common text chunks and when they are changed in the source file, both documents are updated via the result files.

The example code in the file ‘example_source.rnw’ is as follows:

```

1 %<*article|techdoc>
2 %%%%%%%%%%%%%%
3 %% Author: Michael Lundholm
4 %% Email: michael.lundholm@ne.su.se
5 %% Date: 2010-09-06
6 %%%%%%%%%%%%%%
7 %% The project files consists of
8 %% * example_source.rnw (THIS FILE)
9 %% To create other files execute
10 %% R CMD Sweave example_source.rnw
11 %% latex example.ins
12 %% pdflatex example_techdoc.tex
13 %% bibtex example_techdoc
14 %% pdflatex example_article.tex
15 %% bibtex example_article
16 %%%%%%%%%%%%%%
17 \documentclass{article}
18 \usepackage{Sweave, amsmath, natbib}
19 %</article|techdoc>
20 %<article>\title{Article}
21 %<techdoc>\title{Technical documentation}
22 %<article|techdoc>
23 \author{M. Lundholm}
24 \begin{document}
25 \maketitle
26 This note replicates the \citet[p. 56ff]{Kleiber08} estimation of a price per citation elasticity of library subscriptions for economics journals equal to $\input{coef.txt}$.
27 estimation of a price per citation elasticity of
28 library subscriptions for economics journals equal
29 to $\input{coef.txt}$.
30 %</article|techdoc>
31 %<*techdoc>
32 The data, available in R package AER on CRAN, is loaded:
33 <<Loading data>>=
34 data("Journals", package="AER")
35 @
36 %</techdoc>
37 %<*article|techdoc>
38 The data set includes the number of library
39 subscriptions ( $S_i$ ), the number of citations
40 ( $C_i$ ) and the subscription price for libraries
41 ( $P_i$ ). We want to estimate the model
42 
$$\log(S_i) = \alpha_0 + \alpha_1 \log(P_i/C_i) + \epsilon_i$$

43 where  $P_i/C_i$  is the price per citation.
44 %</article|techdoc>
45 %<*techdoc>
46 We then define the price per citation, include
47 the variable in the data frame \texttt{Journals}
48 <<Define variable>>=
49 Journals$citeprice <- Journals$price/Journals$citations
50 @
51 and estimate the model:
52 <<Estimate>>=
53 result <- lm(log(subs) ~ log(citeprice), data=Journals)
54 @
55 %</techdoc>
56 %<*article|techdoc>
57 The result with OLS standard errors is in
58 Table~\ref{ta:est}.
59 <<Result, results=tex, echo=FALSE>>=
60 library(xtable)
61 xtable(summary(result), label="ta:est",
62 caption="Estimation results")
63 @
64 <<echo=FALSE>>=
65 write(round(coef(result)[[2]], 2), file="coef.txt")
66 @
67 \bibliographystyle{abbrvnat}
68 \bibliography{example}
69 \end{document}
70 %</article|techdoc>
71 %<*bib>
72 @Book{ Kleiber08,
73 author = {Christian Kleiber and Achim Zeileis},
74 publisher = {Springer},
75 year = {2008},
76 title = {Applied Econometrics with {R}}}
77 %</bib>
78 %<*dump>
79 <<Write DOCSTRIP installation file>>=
80 writeLines(
81 "\\" \input docstrip.tex
82 \\keepsilent
83 \\askforoverwritefalse
84 \\nopreamble
85 \\nopostamble
86 \\generate{
87 \\file{example_article.tex}%
88 {\\from{example_source.tex}{article}}
89 \\file{example_techdoc.tex}%
90 {\\from{example_source.tex}{techdoc}}
91 \\file{example.bib}%
92 {\\from{example_source.tex}{bib}}}
93 \\endbatchfile
94 ", con="example.ins")
95 @
96 %</dump>
97 
```

The compendium source file contains the following DOCSTRIP tags (for their association to files, see below):

- ‘article’ associated with ‘example_article.tex’, which contains the code to the “advertisement” article,
- ‘techdoc’ associated with ‘example_techdoc.tex’, which contains the code to the technical documentation,
- ‘bib’ associated with ‘example.bib’ which contains the code to the Bib \TeX data base file,
- ‘dump’ associated with no file.

Note that the tags ‘article’ and ‘techdoc’ overlap with each other but not with ‘bib’ and ‘dump’, which in turn are mutually exclusive. There is no untagged material.

Lines 2 – 15 contain general information about the distributed project, which could be more or less elaborate. Here it just states that the project is distributed as a single source file and how the compendium source file should be processed to get the relevant output

Article

M. Lundholm

September 15, 2010

This note replicates the Kleiber and Zeileis [2008, p. 56ff] estimation of a price per citation elasticity of library subscriptions for economics journals equal to -0.53 . The data set includes the number of library subscriptions (S_i), the number of citations (C_i) and the subscription price for libraries (P_i). We want to estimate the model

$$\log(S_i) = \alpha_0 + \alpha_1 \log(P_i/C_i) + \epsilon_i,$$

where P_i/C_i is the price per citation. The result with OLS standard errors is in Table 1.

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	4.7662	0.0559	85.25	0.0000
$\log(\text{citeprice})$	-0.5331	0.0356	-14.97	0.0000

Table 1: Estimation results

References

C. Kleiber and A. Zeileis. *Applied Econometrics with R*. Springer, 2008.

Figure 1: ‘example_article.pdf’.

‘example_article.pdf’ and ‘example_techdoc.pdf’.

When the instructions are followed, `Sweave` is run first on ‘example_source.rnw’ creating the file ‘example_source.tex’, in which the `Sweave` code chunks are replaced by the corresponding R output code wrapped with `LATEX` typesetting commands. One of the R functions used in this `Sweave` session is `writeLines()` (see the lines 80 – 96) so that the `DOCSTRIP` installation file ‘example.ins’ is created before `DOCSTRIP` is run.

This file ‘example_source.tex’ is the `DOCSTRIP` source file from which the `DOCSTRIP` utility, together with the installation file ‘example.ins’, creates the result files ‘example_article.tex’, ‘example_techdoc.tex’ and ‘example.bib’. The two first result files share some but not all code from the `DOCSTRIP` source file. The result files are then run with the `LATEX` family of software (here `pdflatex` and `BibTeX`) to create two PDF-files ‘example_article.pdf’ and ‘example_techdoc.pdf’. These are shown in Figures 1–2.

Note that the entire bibliography (`BibTeX`) file is included on lines 73 – 77 and extracted with `DOCSTRIP`. Note also on line 73 that unless the @ indicating a new bibliographical entry is *not* in column 1 it is mixed up by `Sweave` as a new text chunk and will be removed, with errors as the result when `BibTeX` is run.⁸

The bibliography database file is common to both ‘example_article.tex’ and ‘example_techdoc.tex’. Here the documents have the same single reference. But in

real implementations bibliographies would probably not overlap completely. This way handling references is then preferable since all bibliographical references occur only once in the source file.⁹

In `LaTeX` cross references are handled by writing information to the auxiliary file, which is read by later `LaTeX` runs. This handles references to an object located both before and after the reference in the `LaTeX` file. In `Sweave` ” can be used to refer to R objects created before but not after the reference is made. This is not exemplified here. But since `Sweave` and `LaTeX` are run sequentially an object can be created by R, written to a file (see the code chunk on lines 65 – 67) and then be used in the `LaTeX` run with the command ‘\input{}’ (see code line 29).

Final comments

By making use of combinations of `DOCSTRIP` and (say) ‘`writeLines()`’ and by changing the order in which `Sweave` and `DOCSTRIP` are executed the applications can be made more complex. Such examples may be found Lundholm (2010a,b).¹⁰ Also, the use of `DOCSTRIP` can facilitate the creation of R packages as exemplified by the R data package `sifds` available on CRAN (Lundholm, 2010c). Another type of example would be teaching material, where this article may itself serve as an example. Apart from the `DOCSTRIP`

⁸The tag ‘dump’ is a safeguard against that this material is allocated to some result file by `DOCSTRIP`; in this case to the `BibTeX` data base file.

⁹One alternative would be to replace the command ‘\bibliography{example}’ on line 69 with the content of ‘example_article.bbl’ and ‘example_techdoc.bbl’ appropriately tagged for `DOCSTRIP`. However, this procedure would require an “external” bibliography data base file. The problem then is that each time the data base is changed, manual updating of the parts of ‘example_source.rnw’ that creates ‘example_article.bbl’ and ‘example_techdoc.bbl’ is required. Creating the bibliography data base file via `DOCSTRIP` makes this manual updating unnecessary.

¹⁰An early attempt to implement the ideas presented in this article can be found in Arai et al. (2009).

Technical documentation

M. Lundholm

September 15, 2010

This note replicates the Kleiber and Zeileis [2008, p. 56ff] estimation of a price per citation elasticity of library subscriptions for economics journals equal to -0.53. The data, available in R package AER on CRAN, is loaded:

```
> data("Journals", package = "AER")
```

The data set includes the number of library subscriptions (S_i), the number of citations (C_i) and the subscription price for libraries (P_i). We want to estimate the model

$$\log(S_i) = \alpha_0 + \alpha_1 \log(P_i/C_i) + \epsilon_i,$$

where P_i/C_i is the price per citation. We define the price per citation, include the variable in the data frame Journals

```
> Journals$citeprice <- Journals$price/Journals$citations
```

and estimate the model:

```
> result <- lm(log(subs) ~ log(citeprice), data = Journals)
```

The result with OLS standard errors is in Table 1.

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	4.7662	0.0559	85.25	0.0000
log(citeprice)	-0.5331	0.0356	-14.97	0.0000

Table 1: Estimation results

References

C. Kleiber and A. Zeileis. *Applied Econometrics with R*. Springer, 2008.

Figure 2: ‘example_techdoc.pdf’.

installation file and a Bash script file all code used to produce this article is contained in a single source file. The Bash script, together with DOCSTRIP, creates all example files including the final PDF-files; that is, all example code is executed every time this article is updated. So, if the examples are changed an update of the article via the Bash script also updates the final PDF-files in Figures 1–2.¹¹

Colophone

This article was written on a i486-pc-linux-gnu platform using R version 2.11.1 (2010-05-31), L^AT_EX2 _{ϵ} (2005/12/01) and DOCSTRIP 2.5d (2005/07/29).

Acknowledgement

The compendium implementation presented here is partially developed in projects joint with Mahmood Arai, to whom I am owe several constructive comments on a previous version.

Bibliography

M. Arai, J. Karlsson, and M. Lundholm. On fragile grounds: A replication of Are Muslim immi-

grants different in terms of cultural integration? Accepted for publication in the Journal of the European Economic Association, 2009. URL <http://www.eeassoc.org/index.php?site=JEEA&page=55>.

J. B. Buckheit and D. L. Donoho. WaveLab and reproducible research. In A. Antoniadis and G. Oppenheim, editors, *Wavelets and statistics*, Lecture notes in statistics 103, pages 55–81. Springer Verlag, 1995.

R. Gentleman. Reproducible research: A bioinformatics case study. 2005. URL <http://www.bioconductor.org/docs/papers/2003/Compendium/Golub.pdf>.

R. Gentleman and D. T. Lang. Statistical analyses and reproducible research. *Journal of Computational and Graphical Statistics*, 16:1–23, 2007. URL <http://pubs.amstat.org/doi/pdfplus/10.1198/106186007X178663>.

M. Goossens, F. Mittelbach, and A. Samarin. *The L^AT_EX Companion*. Addison-Wesley, Reading, MA, USA, second edition, 1994.

T. Hothorn. Praktische aspekte der reproduzierbarkeit statistischer analysen in klinischen studien. Institut für Medizininformatik, Biometrie und Epidemiologie, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2006. URL <http://www.imbe.med.uni-erlangen.de/~hothorn/talks/AV.pdf>.

¹¹The compendium source files of projects mentioned in this paragraph, including this article, can be found at <http://people.su.se/~lundh/projects/>.

- F. Leisch. *Sweave: Dynamic generation of statistical reports using literate data analysis*. In W. Härdle and y. . . Bernd Rönz, pages = 575–580, editors, *Compstat 2002 – Proceedings in Computational Statistics*.
- F. Leisch. *Sweave User Manual*, 2008. URL <http://www.stat.uni-muenchen.de/~leisch/Sweave/Sweave-manual.pdf>. R version 2.7.1.
- M. Lundholm. Are inflation forecasts from major swedish forecasters biased? Research paper in economics 2010:10, Department of Economics, Stockholm University, 2010a. URL http://swopec.hhs.se/sunrpe/abs/sunrpe2010_0010.htm.
- M. Lundholm. Sveriges riksbank's inflation interval forecasts 1999–2005. Research paper in economics 2010:11, Department of Economics, Stockholm University, 2010b. URL http://swopec.hhs.se/sunrpe/abs/sunrpe2010_0010.htm.
- M. Lundholm. *sifds: Swedish inflation forecast data set*, 2010c. URL <http://www.cran.r-project.org/web/packages/sifds/index.html>. R package version 0.9.
- E. Meredith and J. S. Racine. Towards reproducible econometric research: The Sweave framework. *Journal of Applied Econometrics*, 2008. URL <http://dx.doi.org/10.1002/jae.1030>. Published Online: 12 Nov 2008.
- F. Mittelbach, D. Duchier, J. Braams, M. Woliński, and M. Wooding. The DOCSTRIP program. Version 2.5d, 2005. URL <http://tug.ctan.org/tex-archive/macros/latex/base/>.
- B. Rising. Reproducible research: Weaving with Stata. StataCorp LP, 2008. URL http://www.stata.com/meeting/italy08/rising_2008.pdf.

*Michael Lundholm
Department of Economics
Stockholm University
SE-106 91 Stockholm
Sweden
michael.lundholm@ne.su.se*

Watch Your Spelling!

by Kurt Hornik and Duncan Murdoch

Abstract We discuss the facilities in base R for spell checking via Aspell, Hunspell or Ispell, which are useful in particular for conveniently checking the spelling of natural language texts in package Rd files and vignettes. Spell checking performance is illustrated using the Rd files in package **stats**. This example clearly indicates the need for a domain-specific statistical dictionary. We analyze the results of spell checking all Rd files in all CRAN packages and show how these can be employed for building such a dictionary.

R and its add-on packages contain large amounts of natural language text (in particular, in the documentation in package Rd files and vignettes). This text is useful for reading by humans and as a testbed for a variety of natural language processing (NLP) tasks, but it is not always spelled correctly. This is not entirely unsurprising, given that available spell checkers are not aware of the special Rd file and vignette formats, and thus rather inconvenient to use. (In particular, we are unaware of ways to take advantage of the ESS (Rossini et al., 2004) facilities to teach Emacs to check the spelling of vignettes by only analyzing the L^AT_EX chunks in suitable T_EX modes.)

In addition to providing facilities to make spell checking for Rd files and vignettes more convenient, it is also desirable to have programmatic (R-level) access to the possibly mis-spelled words (and suggested corrections). This will allow their inclusion into automatically generated reports (e.g., by R CMD check), or aggregation for subsequent analyses. Spell checkers typically know how to extract words from text employing language-dependent algorithms for handling morphology, and compare the extracted words against a known list of correctly spelled ones, the so-called dictionary. However, NLP resources for statistics and related knowledge domains are rather scarce, and we are unaware of suitable dictionaries for these. The result is that domain-specific terms in texts from these domains are typically reported as possibly mis-spelled. It thus seems attractive to create additional dictionaries based on determining the most frequent possibly mis-spelled terms in corpora such as the Rd files and vignettes in the packages in the R repositories.

In this paper, we discuss the spell check functionality provided by `aspell()` and related utilities made available in the R standard package **utils**. After a quick introduction to spell checking, we indicate how `aspell()` can be used for checking individual files and packages, and how this can be integrated into typical work flows. We then compare the agreement of the spell check results obtained by two different

programs (Aspell and Hunspell) and their respective dictionaries. Finally, we investigate the possibility of using the CRAN package Rd files for creating a statistics dictionary.

Spell checking

The first spell checkers were widely available on mainframe computers in the late 1970s (e.g., http://en.wikipedia.org/wiki/Spell_checker). They actually were “verifiers” instead of “correctors”, reporting words not found in the dictionary but not making useful suggestions for replacements (e.g., as close matches in the Levenshtein distance to terms from the dictionary). In the 1980s, checkers became available on personal computers, and were integrated into popular word-processing packages like WordStar. Recently, applications such as web browsers and email clients have added spell check support for user-written content. Extending coverage from English to beyond western European languages has required software to deal with character encoding issues and increased sophistication in the morphology routines, particularly with regard to heavily-agglutinative languages like Hungarian and Finnish, and resulted in several generations of open-source checkers. Limitations of the basic unigram (single-word) approach have led to the development of context-sensitive spell checkers, currently available in commercial software such as e.g. Microsoft Office 2007. Another approach to spell checking is using adaptive domain-specific models for mis-spellings (e.g., based on the frequency of word *n*-grams), as employed for example in web search engines (“Did you mean ...?”).

The first spell checker on Unix-alikes was `spell` (originally written in 1971 in PDP-10 Assembly language and later ported to C), which read an input file and wrote possibly mis-spelled words to output (one word per line). A variety of enhancements led to (International) Ispell (<http://lasr.cs.ucla.edu/geoff/ispell.html>), which added interactivity (hence “i”-spell), suggestion of replacements, and support for a large number of European languages (hence “international”). It pioneered the idea of a programming interface, originally intended for use by Emacs, and awareness of special input file formats (originally, T_EX or nroff/troff).

GNU Aspell, usually called just Aspell (<http://aspell.net>), was mainly developed by Kevin Atkinson and designed to eventually replace Ispell. Aspell can either be used as a library (in fact, the Omegahat package **Aspell** (Temple Lang, 2005) provides a fine-grained R interface to this) or as a standalone program. Compared to Ispell, Aspell can also easily check documents in UTF-8 without having to use a special dictionary, and supports using multiple dictionaries. It also

tries to do a better job suggesting corrections (Ispell only suggests words with a Levenshtein distance of 1), and provides powerful and customizable TeX filtering. Aspell is the standard spell checker for the GNU software system, with packages available for all common Linux distributions. E.g., for Debian/Ubuntu flavors, **aspell** contains the programs, **aspell-en** the English language dictionaries (with American, British and Canadian spellings), and **libaspell-dev** provides the files needed to build applications that link against the Aspell libraries.

See <http://aspell.net> for information on obtaining Aspell, and available dictionaries. A native Windows build of an old release of Aspell is available on <http://aspell.net/win32/>. A current build is included in the Cygwin system at <http://cygwin.com>¹, and a native build is available at <http://www.ndl.kiev.ua/content/patch-aspell-0605-win32-compilation>.

Hunspell (<http://hunspell.sourceforge.net>) is a spell checker and morphological analyzer designed for languages with rich morphology and complex word compounding or character encoding, originally designed for the Hungarian language. It is in turn based on Myspell, which was started by Kevin Hendricks to integrate various open source spelling checkers into the OpenOffice.org build, and facilitated by Kevin Atkinson. Unlike Myspell, Hunspell can use Unicode UTF-8-encoded dictionaries. Hunspell is the spell checker of OpenOffice.org, Mozilla Firefox 3 & Thunderbird and Google Chrome, and it is also used by proprietary software like MacOS X. Its TeX support is similar to Ispell, but nowhere near that of Aspell. Again, Hunspell is conveniently packaged for all common Linux distributions (with Debian/Ubuntu packages **hunspell** for the standalone program, dictionaries including **hunspell-en-us** and **hunspell-en-ca**, and **libhunspell-dev** for the application library interface). For Windows, the most recent available build appears to be version 1.2.8 on <http://sourceforge.net/projects/hunspell/files>.

Aspell and Hunspell both support the so-called Ispell pipe interface, which reads a given input file and then, for each input line, writes a single line to the standard output for each word checked for spelling on the line, with different line formats for words found in the dictionary, and words not found, either with or without suggestions. Through this interface, applications can easily gain spell-checking capabilities (with Emacs the longest-serving “client”).

Spell checking with R

The basic function for spell checking provided by the **utils** package is **aspell()**, with synopsis

```
aspell(files, filter, control = list(),
       encoding = "unknown", program = NULL)
```

Argument **files** is a character vector with the names of the files to be checked (in fact, in R 2.12.0 or later alternatively a list of R objects representing connections or having suitable **srcrefs**), **control** is a list or character vector of control options (command line arguments) to be passed to the spell check program, and **program** optionally specifies the name of the program to be employed. By default, the system path is searched for **aspell**, **hunspell** and **ispell** (in that order), and the first one found is used. Encodings which can not be inferred from the files can be specified via **encoding**. Finally, one can use argument **filter** to specify a filter for processing the files before spell checking, either as a user-defined function, or a character string specifying a built-in filter, or a list with the name of a built-in filter and additional arguments to be passed to it. The built-in filters currently available are "**Rd**" and "**Sweave**", corresponding to functions **RdTextFilter** and **SweaveTeXFilter** in package **tools**, with self-explanatory names: the former blanks out all non-text in an **Rd** file, dropping elements such as **\email** and **\url** or as specified by the user via argument **drop**; the latter blanks out code chunks and Noweb markup in an **Sweave** input file.

aspell() returns a data frame inheriting from "**aspell**" with the information about possibly misspelled words, which has useful **print()** and **summary()** methods. For example, consider '**lm.Rd**' in package **stats** which provides the documentation for fitting linear models. Assuming that **src** is set to the URL for the source of that file, i.e. "<http://svn.R-project.org/R/trunk/src/library/stats/man/lm.Rd>", and **f** is set to "**lm.Rd**", we can obtain a copy and check it as follows:

```
> download.file(src, f)
> a <- aspell(f, "Rd")
> a

accessor
  lm.Rd:128:25

ANOVA
  lm.Rd:177:7

datasets
  lm.Rd:199:37

differenced
  lm.Rd:168:57

formulae
  lm.Rd:103:27

Ihaka
  lm.Rd:214:68

logicals
  lm.Rd:55:26

priori
  lm.Rd:66:56
```

¹The Cygwin build requires Unix-style text file inputs, and Cygwin-style file paths, so is not fully compatible with **aspell()** in R.

```
regressor
lm.Rd:169:3
```

(results will be quite different if one forgets to use the Rd filter). The output is self-explanatory: for each possibly mis-spelled word, we get the word and the occurrences in the form `file:linenum:colnum`. Clearly, terms such as ‘ANOVA’ and ‘regressor’ are missing from the dictionary; if one was not sure about ‘datasets’, one could try visiting <http://en.wiktionary.org/wiki/datasets>.

If one is only interested in the possibly mis-spelled words themselves, one can use

```
> summary(a)
Possibly mis-spelled words:
[1] "accessor"      "ANOVA"        "datasets"
[4] "differenced"   "formulae"     "Ihaka"
[7] "logicals"       "priori"       "regressor"
```

(or directly access the `Original` variable); to see the suggested corrections, one can print with `verbose = TRUE`:

```
> print(subset(a,
+                 Original %in%
+                 c("ANOVA", "regressor")),
+       verbose = TRUE)
```

```
Word: ANOVA (lm.Rd:177:7)
Suggestions: AN OVA AN-OVA NOVA ANICA ANIA
NOVAE ANA AVA INVAR NOV OVA UNIV ANITA
ANNORA AVIVA ABOVE ENVOY ANNA NEVA ALVA
ANYA AZOV ANON ENVY JANEVA ANODE ARNO
IVA ANVIL NAIVE OLVA ANAL AVON AV IONA
NV NAVY OONA AN AVOW INFO NAVE ANNOY
ANN ANY ARV AVE EVA INA NEV ONO UNA
ANCHOVY ANA'S
```

```
Word: regressor (lm.Rd:169:3)
Suggestions: regress or regress-or regress
            regressive regressed regresses
            aggressor regressing egress regrets
            Negress redress repress recross regrows
            egress's Negress's
```

Note that the first suggestion is ‘regress or’.

The print method also has an `indent` argument controlling the indentation of the locations of possibly mis-spelled words. The default is 2; Emacs users may find it useful to use an indentation of 0 and visit output (directly when running R from inside ESS, or redirecting output to a file using `sink()` in grep-mode).

The above spell check results were obtained using Aspell with an American English dictionary. One can also use several dictionaries:

```
> a2 <- aspell(f, "Rd",
+               control =
+               c("--master=en_US",
+                  "--add-extra-dicts=en_GB"))
> summary(a2)
```

Possibly mis-spelled words:

```
[1] "accessor"      "ANOVA"        "datasets"
[4] "differenced"   "formulae"     "Ihaka"
[7] "priori"        "regressor"
> setdiff(a$Original, a2$Original)
[1] "formulae"
```

This shows that Aspell no longer considers ‘formulae’ mis-spelled when the “British” dictionary is added, and also exemplifies how to use the `control` argument: using Hunspell, the corresponding choice of dictionaries could be achieved using `control = "-d en_US,en_GB"`. (Note that the dictionaries differ, as we shall see below). This works for “system” dictionaries; one can also specify “personal” dictionaries (using the common command line option ‘`-p`’).

We already pointed out that Aspell excels in its `\TeX` filtering capabilities, which is obviously rather useful for spell-checking R package vignettes in Sweave format. Similar to Ispell and Hunspell, it provides a `\TeX` mode (activated by the common command line option ‘`-t`’) which knows to blank out `\TeX` commands. In addition to the others, it allows to selectively blank out options and parameters of these commands. This works via specifications of the form ‘`name signature`’ where the first gives the name of the command and the second is a string containing ‘`p`’ or ‘`o`’ indicating to blank out parameters or options, or ‘`P`’ or ‘`O`’ indicating to keep them for checking. E.g., ‘`foo Pop`’ specifies to check the first parameter and then skip the next option and the second parameter (even if the option is not present), i.e. following the pattern `\foo{checked} [unchecked] {unchecked}`, and is passed to Aspell as

```
--add-tex-command="foo Pop"
```

Aspell’s `\TeX` filter is already aware of a number of common (`\LaTeX`) commands (but not, e.g., of `\citet` used for parenthetical citations using `natbib`). However, this filter is based on C++ code internal to the Aspell data structures, and hence not reusable for other spell checkers: we are currently exploring the possibility to provide similar functionality using R.

Package `utils` also provides three additional utilities (exported in R 2.12.0 or later):

```
aspell_package_Rd_files(),
aspell_package_vignettes(), and
aspell_write_personal_dictionary_file().
```

The first two, obviously for spell checking the Rd files and (Sweave) vignettes in a package, are not only useful for automatically computing all relevant files and choosing the appropriate filter (and, in the case of vignettes, adding a few commands like `\Sexpr` and `\citet` to the blank out list), but also because they support a package default mechanism described below. To see a simple example, we use Aspell to check the spelling of all Rd files in package `stats`:

```
> require("tools")
> drop <- c("\\author", "\\source", "\\references")
> ca <- c("--master=en_US",
+           "--add-extra-dicts=en_GB")
> asa <- aspell_package_Rd_files("stats",
+                                 drop = drop, program = "aspell",
+                                 control = ca)
```

This currently (2011-05-11) finds 1114 possibly misspelled words which should hopefully all be false positives (as we regularly use `aspell()` to check the spelling in R's Rd files). The most frequent occurrences are

```
> head(sort(table(asa$Original), decreasing = TRUE))

quantile dendrogram      AIC univariate
      57          44        42          42
quantiles     ARIMA       30          23
```

which should clearly be in every statistician's dictionary!

Package defaults for spell checking (as well as a personal dictionary file) can be specified in a '`defaults.R`' file in the '`.aspell`' subdirectory of the top-level package source directory, and are provided via assignments to suitably named lists, as e.g.

```
vignettes <-
  list(control = "--add-tex-command=mycmd op")
```

for vignettes and assigning to `Rd_files` for Rd files defaults. For the latter, one can also give a `drop` default specifying additional Rd sections to be blanked out, e.g.,

```
Rd_files <- list(drop = c("\\author",
                           "\\source",
                           "\\references"))
```

The default lists can also contain a personal element for specifying a package-specific personal dictionary. A possible work flow for package maintainers is the following: one runs the spell checkers and corrects all true positives, leaving the false positives. Obviously, these should be recorded so that they are not reported the next time when texts are modified and checking is performed again. To do this one re-runs the check utility (e.g., `aspell_package_vignettes()` and saves its results into a personal dictionary file using `aspell_write_personal_dictionary_file()`). This file is then moved to the '`.aspell`' subdirectory (named, e.g., '`vignettes.pws`') and then activated via the `defaults` file using, e.g.,

```
vignettes <-
  list(control = "--add-tex-command=mycmd op",
       personal = "vignettes.pws")
```

Following a new round of checking and correcting, the procedure is repeated (with the personal dictionary file temporarily removed before re-running the utility and re-creating the dictionary file; currently,

there is no user-level facility for reading/merging personal dictionaries).

A different work flow is to keep the previous spell check results and compare the current one to the most recent previous one. In fact, one of us (KH) uses this approach to automatically generate "check diffs" for the R Rd files, vignettes and Texinfo manuals and correct the true positives found.

A spell check utility R itself does not (yet) provide is one for checking a list of given words, which is easily accomplished as follows:

```
> aspell_words <- function(x, control = list()) {
+   con <- textConnection(x)
+   on.exit(close(con))
+   aspell(list(con), control = control)
+ }
```

Of course, this reports useless locations, but is useful for programmatic dictionary lookup:

```
> print(aspell_words("flavour"), verbose = TRUE)

Word: flavour (<unknown>:1:1)
Suggestions: flavor flavors favor flour
               flair flours floury Flor flavor's Fleur
               floor flyover flour's
```

(again using Aspell with an American English dictionary).

Spell checking performance

Should one use Aspell or Hunspell (assuming convenient access to both)? Due to its more powerful morphological algorithms, Hunspell is slower. For vignettes, Aspell performance is clearly superior, given its superior TeX capabilities. For Rd files, we compare the results we obtain for package `stats`. Similar to the above, we use Hunspell on these Rd files:

```
> ch <- "-d en_US,en_GB"
> ash <- aspell(files, filter,
+                  program = "hunspell", control = ch)
```

(In fact, this currently triggers a segfault on at least Debian GNU/Linux squeeze on file 'lowess.Rd' once the `en_GB` dictionary is in place: so we really check this file using `en_US` only, and remove two GB spellings.) We then extract the words and terms (unique words) reported as potentially mis-spelled:

```
> asaw <- asa$Original; asat <- unique(asaw)
> ashw <- ash$Original; asht <- unique(ashw)
> allt <- unique(c(asat, asht))
```

This gives 1114 and 355 words and terms for Aspell, and 789 and 296 for Hunspell, respectively. Cross-tabulation yields

```
> tab <- table(Aspell = allt %in% asat,
+              Hunspell = allt %in% asht)
> tab
```

```
Hunspell
Aspell FALSE TRUE
  FALSE    0   27
  TRUE    86  269
```

and the amount of agreement can be measured via the Jaccard dissimilarity coefficient, which we compute “by hand” via

```
> sum(tab[row(tab) != col(tab)]) / sum(tab)
[1] 0.2958115
```

which is actually quite large. To gain more insight, we inspect the most frequent terms found only by Aspell

```
> head(sort(table(asaw[! asaw %in% asht]),
+           decreasing = TRUE), 4L)
quantile univariate quantiles      th
  57          42          30        18
```

and similarly for Hunspell:

```
> head(sort(table(ashw[! ashw %in% asat]),
+           decreasing = TRUE), 4L)
's Mallows'  hessian      BIC
  21          6          5        4
```

indicating that Aspell currently does not know quantiles, and some differences in the morphological handling (the apostrophe s terms reported by Hunspell are all instances of possessive forms of words typeset using markup blanked out by the filtering). It might appear that Hunspell has a larger and hence “better” dictionary: but in general, increasing dictionary size will also increase the true negative rate. To inspect commonalities, we use

```
> head(sort(table(asaw[asaw %in%
+                  intersect(asat, asht)]),
+           decreasing = TRUE),
+           12L)
dendrogram      AIC      ARIMA      ARMA
  44          42          23          22
Wilcoxon      Tukey's      GLM       Nls
  16          15          12          10
periodogram    MLE      GLMs      IWLS
  10          9           8           8
```

re-iterating the fact that a statistics dictionary is needed.

We can also use the above to assess the “raw” performance of the spell checkers, by counting the number of terms and words in the stats Rd file corpus. Using the text mining infrastructure provided by package **tm** (Feinerer et al., 2008), this can be achieved as follows:

```
> require("tm")
> texts <-
+   lapply(files, RdTextFilter, drop = drop)
```

```
> dtm <- DocumentTermMatrix(
+   Corpus(VectorSource(texts)),
+   control = list(removePunctuation = TRUE,
+                 removeNumbers = TRUE,
+                 minWordLength = 2L))
```

(the **control** argument is chosen to match the behavior of the spell checkers). This gives

```
> dtm
A document-term matrix (304 documents, 3533 terms)
Non-/sparse entries: 31398/1042634
Sparsity : 97%
Maximal term length: 24
> sum(dtm)
[1] 69910
```

with 3533 terms and 69910 words, so that Aspell would give “raw” false positive rates of 10.05 and 1.59 percent for terms and words, respectively (assuming that the regular spell checking of the Rd files in the R sources has truly eliminated all mis-spellings).

Towards a dictionary for statistics

Finally, we show how the spell check results can be employed for building a domain-specific dictionary (supplementing the default ones). Similar to the above, we run Aspell on all Rd files in CRAN and base R packages (Rd files are typically written in a rather terse and somewhat technical style, but should nevertheless be a good proxy for current terminology in computational statistics). With results in **ac**, we build a corpus obtained by splitting the words according to the Rd file in which they were found, and compute its document-term matrix:

```
> terms <- ac$Original
> files <- ac$file
> dtm_f <- DocumentTermMatrix(
+   Corpus(VectorSource(split(terms, files))),
+   control =
+   list(tolower = FALSE,
+        minWordLength = 2L))
```

(Again, **minWordLength = 2L** corresponds to the Aspell default to ignore single-character words only.) This finds 50658 possibly mis-spelled terms in 43287 Rd files, for a total number of 332551 words. As is quite typical in corpus linguistics, the term frequency distribution is heavily left-tailed

```
> require("slam")
> tf <- col_sums(dtm_f)
> summary(tf)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1.000	1.000	2.000	6.565	4.000	3666.000

and can conveniently be visualized by plotting the cumulative frequencies for the terms sorted in decreasing order (see Figure 1):

```
> tf <- sort(tf, decreasing = TRUE)
> par(las=1)
> plot(seq_along(tf),
+       cumsum(tf) / sum(tf),
+       type = "l",
+       xlab = "Number of most frequent terms",
+       ylab = "Proportion of words",
+       panel.first = abline(v=1000, col="gray"))
```

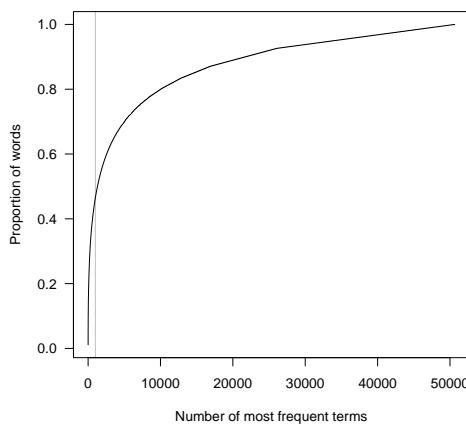


Figure 1: The number of possibly mis-spelled words (cumulative term frequency) in the Rd files in the CRAN and R base packages versus the number of terms (unique words), with terms sorted in decreasing frequency. The vertical line indicates 1000 terms.

We see that the 1000 most frequent terms already cover about half of the words. The corpus also reasonably satisfies Heap's Law (e.g., http://en.wikipedia.org/wiki/Heaps%27_law or Manning et al. (2008)), an empirical law indicating that the vocabulary size V (the number of different terms employed) grows polynomially with text size T (the number of words), as shown in Figure 2:

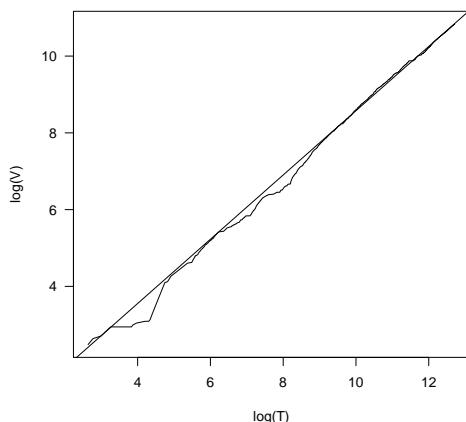


Figure 2: An illustration of Heap's Law for the corpus of possibly mis-spelled words in the Rd files in the CRAN and R base packages, taking the individual files as documents.

```
> Heaps_plot(dtm_f)

(Intercept)           x
0.2057821   0.8371087
```

(the regression coefficients are for the model $\log(V) = \alpha + \beta \log(T)$).

To develop a dictionary, the term frequencies may need further normalization to weight their "importance". In fact, to adjust for possible authoring and package size effects, it seems preferable to aggregate the frequencies according to package. We then apply a simple, so-called binary weighting scheme which counts occurrences only once, so that the corresponding aggregate term frequencies become the numbers of packages the terms occurred in. Other weighting schemes are possible (e.g., normalizing by the total number of words checked in the respective package). This results in term frequencies tf with the following characteristics:

```
> summary(tf)

      Min. 1st Qu. Median     Mean 3rd Qu.    Max.
1.000   1.000   1.000   2.038   1.000 682.000

> quantile(tf, c(seq(0.9, 1.0, by = 0.01)))

 90% 91% 92% 93% 94% 95% 96% 97% 98% 99%
 3    3    3    4    4    4    5    6    7   10   17
100%
682
```

Again, the frequency distribution has a very heavy left tail. We apply a simple selection logic, and drop terms occurring in less than 0.5% of the packages (currently, about 12.5), leaving 764 terms for possible inclusion into the dictionary. (This also eliminates the terms from the few CRAN packages with non-English Rd files.) For further processing, it is highly advisable to take advantage of available lexical resources. One could programmatically employ the APIs of web search engines: but note that the spell correction facilities these provide are not domain specific. Using WordNet (Fellbaum, 1998), the most prominent lexical database for the English language, does not help too much: it only has entries for about 10% of our terms.

We use the already mentioned Wiktionary (<http://www.wiktionary.org/>), a multilingual, web-based project to create a free content dictionary which is run by the Wikimedia Foundation (like its sister project Wikipedia) and which has successfully been used for semantic web tasks (e.g., Zesch et al., 2008). Wiktionary provides a simple API at <http://en.wiktionary.org/w/api.php> for English language queries. One can look up given terms by queries using parameters `action=query`, `format=xml`, `prop=revision`, `rvprop=content` and `titles` as a list of the given terms collapsed by a vertical bar (actually, a maximum of 50 terms can be looked up in one query). When doing so via R, one can conveniently use the XML package (Temple Lang, 2010) to parse

the result. For our terms, the lookup returns information for 416 terms. However, these can not be accepted unconditionally into the dictionary: in addition to some terms being flagged as mis-spelled or archaic, some terms have possible meanings that were almost certainly not intended (e.g., “wether” as a castrated buck goat or ram). In addition, 2-character terms need special attention (e.g., ISO language codes most likely not intended). Therefore, we extract suitable terms by serially working through suitable subsets of the Wiktionary results (e.g., terms categorized as statistical or mathematical (unfortunately, only a very few), acronyms or initialisms, and plural forms) and inspecting these for possible inclusion. After these structured eliminations, the remaining terms with query results as well as the ones Wiktionary does not know about (the majority of which actually are mis-spellings) are handled. Quite a few of our terms are surnames, and for now we only include the most frequent ones.

```
> p <- readLines("en-stats.pws")
```

We finally obtain a dictionary with 443 terms, which we save into ‘en-stats.pws’ using `aspell_write_personal_dictionary_file()`. We intend to make this easily available from within R at some point in the future.

A few false positives remain systematically: Aspell’s word extraction turns ‘1st’ and ‘2nd’ into ‘st’ and ‘nd’ (and so forth), which clearly are not terms to be included in the dictionary. Similarly, ‘a-priori’ is split with ‘priori’ reported as mis-spelled (actually, Aspell has some support for accepting “run-together words”). These and other cases could be handled by enhancing the filtering routines before calling the spell checkers.

We re-run the checks on `stats` using this dictionary:

```
> asap <-
+   aspell(files, filter, program = "aspell",
+         control = c(ca, "-p ./en-stats.pws"))
```

(the strange ‘./’ is needed to ensure that Aspell does not look for the personal dictionary in its system paths). This reduces the number of possibly mis-spelled words found to 411, and the “estimated” false positive rates for terms and words to 5.97 and 0.59 percent, respectively.

Many of the remaining false positives could also be eliminated by using the appropriate Rd markup (such as `\acronym` or `\code`). With R 2.12.0 or later, one can also use markup in titles. In fact, R 2.12.0 also adds `\newcommand` tags to the Rd format to allow user-defined macros: using, e.g., `\newcommand{\I}{#1}` one could wrap content to be ignored for spell checking into `\I{}`, as currently all user-defined macros are blanked out by the Rd filtering. Similar considerations apply to vignettes when using Aspell’s superior filtering capabilities.

Using a custom dictionary has significantly reduced the false positive rate, demonstrating the

usefulness of the approach. Clearly, more work will be needed: modern statistics needs better lexical resources, and a dictionary based on the most frequent spell check false alarms can only be a start. We hope that this article will foster community interest in contributing to the development of such resources, and that refined domain specific dictionaries can be made available and used for improved text analysis with R in the near future.

Bibliography

- Ingo Feinerer, Kurt Hornik, and David Meyer. Text mining infrastructure in R. *Journal of Statistical Software*, 25(5):1–54, 2 2008. ISSN 1548-7660. URL <http://www.jstatsoft.org/v25/i05>.
- Christiane Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, MA, 1998. ISBN 978-0-262-06197-1.
- Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. URL <http://nlp.stanford.edu/IR-book/information-retrieval-book.html>.
- Anthony J. Rossini, Richard M. Heiberger, Rodney Sparapani, Martin Mächler, and Kurt Hornik. Emacs speaks statistics: A multi-platform, multi-package development environment for statistical analysis. *Journal of Computational and Graphical Statistics*, 13(1):247–261, 2004.
- Duncan Temple Lang. *Interface to aspell library*, 2005. URL <http://www.omegahat.org/Aspell>. R package version 0.2-0.
- Duncan Temple Lang. *XML: Tools for parsing and generating XML within R and S-Plus.*, 2010. URL <http://CRAN.R-project.org/package=XML>. R package version 3.1-1.
- Torsten Zesch, Christof Müller, and Iryna Gurevych. Using wiktionary for computing semantic relatedness. In *AAAI’08: Proceedings of the 23rd national conference on Artificial intelligence*, pages 861–866. AAAI Press, 2008. ISBN 978-1-57735-368-3.
- Kurt Hornik*
Department of Finance, Accounting and Statistics
Wirtschaftsuniversität Wien
Augasse 2–6, 1090 Wien
Austria
Kurt.Hornik@R-project.org
- Duncan Murdoch*
Department of Statistical and Actuarial Sciences
University of Western Ontario
London, Ontario, Canada
murdoch@stats.uwo.ca

Ckmeans.1d.dp: Optimal k -means Clustering in One Dimension by Dynamic Programming

by Haizhou Wang and Mingzhou Song

Abstract

The heuristic k -means algorithm, widely used for cluster analysis, does not guarantee optimality. We developed a dynamic programming algorithm for optimal one-dimensional clustering. The algorithm is implemented as an R package called **Ckmeans.1d.dp**. We demonstrate its advantage in optimality and runtime over the standard iterative k -means algorithm.

Introduction

Cluster analysis offers a useful way to organize and represent complex data sets. It is used routinely for data analysis in fields such as bioinformatics. The k -means problem is to partition data into k groups such that the sum of squared Euclidean distances to each group mean is minimized. However, the problem is NP-hard in a general Euclidean space, even when the number of clusters k is 2 (Aloise et al., 2009; Dasgupta and Freund, 2009), or when the dimensionality is 2 (Mahajan et al., 2009). The standard iterative k -means algorithm (Lloyd, 1982) is a widely used heuristic solution. The algorithm iteratively calculates the within-cluster sum of squared distances, modifies group membership of each point to reduce the within-cluster sum of squared distances, and computes new cluster centers until local convergence is achieved. The time complexity of this standard k -means algorithm is $O(qknp)$, where q is the number of iterations, k is the number of clusters, n is the sample size, and p is the dimensionality (Manning et al., 2008).

However, the disadvantages of various heuristic k -means algorithms in repeatability, optimality, and runtime may limit their usage in medical and scientific applications. For example, when patients are clustered according to diagnostic measurements on expression of marker genes, it can be critical that a patient consistently falls into a same diagnostic group to receive appropriate treatment, regardless of how many times the clustering is applied. In this case, both cluster repeatability and optimality are important. The result of heuristic k -means clustering, heavily dependent on the initial cluster centers, is neither always optimal nor repeatable. Often one restarts the procedure a number of times to mitigate the problem. However, when k is big, the number of restarts for k -means to approach an optimal solution can be prohibitively high and may lead to a substantial increase

in runtime. As clustering is often a preprocessing step in data analysis or modeling, such inadequacy can pose a nuisance for further steps.

In response to this challenge, our objective is to develop a practical and convenient clustering algorithm in R to guarantee optimality in a one-dimensional (1-D) space. The motivation originated from discrete system modeling such as Boolean networks (Kauffman, 1969, 1993) and generalized logical networks (Song et al., 2009), where continuous values must be converted to discrete ones before modeling. Our algorithm follows a comparable dynamic programming strategy used in a 1-D quantization problem to preserve probability distributions (Song et al., 2010), the segmented least squares problem and the knapsack problem (Kleinberg and Tardos, 2006). The objective function in the k -means problem is the sum of squares of within-cluster distances. We present an exact dynamic programming solution with a runtime of $O(n^2k)$ to the 1-D k -means problem.

We implemented the algorithm in the R package **Ckmeans.1d.dp** (Song and Wang, 2011) and evaluated its performance by simulation studies. We demonstrate how the standard k -means algorithm may return unstable clustering results, while our method guarantees optimality. Our method is implemented in C++ to take advantage of the practical speedup of a compiled program. This C++ implementation is further wrapped in an R package so that it can be conveniently called in R.

Optimal 1-D clustering by dynamic programming

We first define the k -means problem. Let x_1, \dots, x_n be an input array of n numbers sorted in non-descending order. The problem of 1-D k -means clustering is defined as assigning elements of the input 1-D array into k clusters so that the sum of squares of within-cluster distances from each element to its corresponding cluster mean is minimized. We refer to this sum as within-cluster sum of squares, or *withinss* for short.

We introduce a dynamic programming algorithm to guarantee optimality of clustering in 1-D. We define a sub-problem as finding the minimum *withinss* of clustering x_1, \dots, x_i into m clusters. We record the corresponding minimum *withinss* in entry $D[i, m]$ of an $n+1$ by $k+1$ matrix D . Thus $D[n, k]$ is the minimum *withinss* value to the original problem. Let j be the index of the smallest number in cluster m in an optimal

solution to $D[i, m]$. It is evident that $D[j - 1, m - 1]$ must be the optimal *withinss* for the first $j - 1$ points in $m - 1$ clusters, for otherwise one would have a better solution to $D[i, m]$. This establishes the optimal substructure for dynamic programming and leads to the recurrence equation

$$D[i, m] = \min_{m \leq j \leq i} \{D[j - 1, m - 1] + d(x_j, \dots, x_i)\}, \\ 1 \leq i \leq n, 1 \leq m \leq k$$

where $d(x_j, \dots, x_i)$ is the sum of squared distances from x_j, \dots, x_i to their mean. The matrix is initialized as $D[i, m] = 0$, when $m = 0$ or $i = 0$.

Using the above recurrence, we can obtain $D[n, k]$, the minimum *withinss* achievable by a clustering of the given data. In the meanwhile, $D[n, m]$ gives the minimum *withinss* if all n numbers are clustered into m groups. By definition each entry requires $O(n^2)$ time to compute using the given recurrence if $d(x_j, \dots, x_i)$ is computed in linear time, resulting in $O(n^3k)$ total runtime. However, $d(x_j, \dots, x_i)$ can be computed in constant time in the recurrence. This is possible because $d(x_j, \dots, x_i)$ can be computed progressively based on $d(x_{j+1}, \dots, x_i)$ in constant time. Using a general index from 1 to i , we iteratively compute

$$d(x_1, \dots, x_i) = d(x_1, \dots, x_{i-1}) + \frac{i-1}{i}(x_i - \mu_{i-1})^2 \\ \mu_i = \frac{x_i + (i-1)\mu_{i-1}}{i}$$

where μ_i is the mean of the first i elements. This iterative computation of $d(x_j, \dots, x_i)$ reduces the overall runtime of the dynamic programming algorithm to $O(n^2k)$.

To find a clustering of data with the minimum *withinss* of $D[n, k]$, we define an auxiliary n by k matrix B to record the index of the smallest number in cluster m :

$$B[i, m] = \operatorname{argmin}_{m \leq j \leq i} \{D[j - 1, m - 1] + d(x_j, \dots, x_i)\}, \\ 1 \leq i \leq n, 1 \leq m \leq k$$

Then we backtrack from $B[n, k]$ to obtain the starting and ending indices for all clusters and generate an optimal solution to the k -means problem. The backtrack is done in $O(k)$ time.

The space complexity of the dynamic programming is $O(nk)$ because we use an $(n + 1) \times (k + 1)$ matrix D to record minimum *withinss* and an $n \times k$ matrix B for backtracking.

Implementation

We implemented this dynamic programming algorithm and created an R package **Ckmeans.1d.dp**. To take advantage of the runtime speed up by a compiled language over R, an interpreted language, we developed the dynamic programming solution in C++.

We compiled the C++ code to a binary dynamic library, which can be called within R through function `Ckmeans.1d.dp()` provided in the package.

Performance evaluation and comparison with the standard k -means

Simulated data sets

We simulated data sets containing clusters using 1-D Gaussian mixture models. In a Gaussian mixture model, the number of components is the true number of clusters. The mean of each Gaussian component is randomly sampled from the uniform distribution from -1 to 1 and the standard deviation of a component is uniformly distributed from 0 to 0.2 .

Optimality

We first compared the optimality of `Ckmeans.1d.dp()` and the standard k -means method on 1-D data. We used the Hartigan and Wong (1979) implementation of k -means, as provided by the `kmeans()` function in R. The core of the `kmeans()` function is also implemented in C/C++. As `Ckmeans.1d.dp()` guarantees optimality of clustering, we define the relative difference from the `kmeans()` clustering result to the optimal value produced by `Ckmeans.1d.dp()` as

$$\frac{\text{withinss}(\text{k-means}) - \text{withinss}(\text{Ckmeans.1d.dp})}{\text{withinss}(\text{Ckmeans.1d.dp})}$$

which measures deviation of the k -means result from the optimal value.

The data sets were randomly generated from Gaussian mixture models with 2 to 50 components. We ran `Ckmeans.1d.dp()` and `kmeans()` on input data given the correct number of clusters k . The relative difference in *withinss* from the `kmeans()` to the optimal value produced by `Ckmeans.1d.dp()` is shown as a function of k in Figure 1.

Although one hopes to get better clusters by restarting k -means several times, the simulation suggests that it still cannot guarantee optimality when there are many clusters in the data. This clearly demonstrates the advantage of `Ckmeans.1d.dp()` in optimality.

Runtime

We then compared the empirical runtime of `Ckmeans.1d.dp()` with the standard k -means method on 1-D data. Using 1-D Gaussian mixture models, we generated five input arrays of size 100, 1,000, 10,000, 100,000, and 1,000,000, respectively. The Gaussian mixture models had 2 to 50 components.

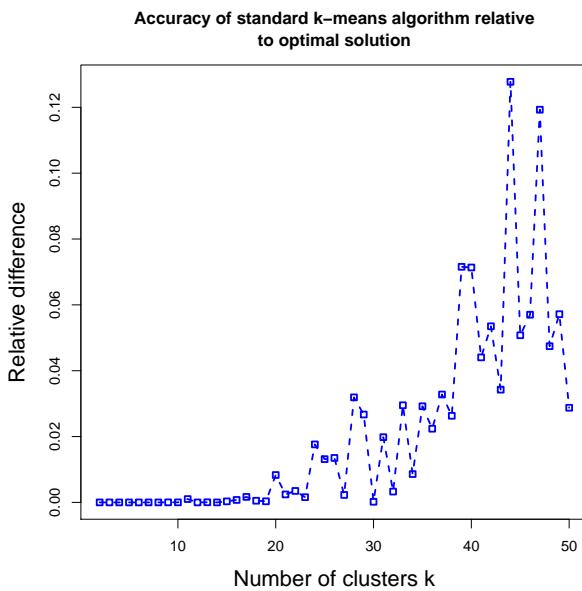


Figure 1: The accuracy of `kmeans()` becomes worse as the number of clusters increases. Accuracy is indicated by the relative difference in `withinss` from `kmeans()` to the optimal value returned by `Ckmeans.1d.dp()`. The input data sets of size 10,000 were sampled from Gaussian mixture models of 2 to 50 components. `kmeans()` was set to restart 20 times for each data set.

We ran both programs on each input array ten times to obtain average empirical runtimes in three different settings. All simulations were run on a desktop computer with an Intel Core i7 860 processor and 8GB memory, running OpenSUSE 11.3 and R 2.11.1.

In the first setting (Figure 2), runtime is obtained as a function of input data size from 100 to 1,000,000 for both `Ckmeans.1d.dp()` and `kmeans()` without constraints on optimality. The number of components in the Gaussian mixture models is fixed to 2.

According to Figure 2, as the input size increases, the runtime of `Ckmeans.1d.dp()` increases faster than the runtime of `kmeans()`. However, the result returned by `kmeans()` may not be optimal (the restart of `kmeans()` was set to 1). Without any constraints on optimality, `kmeans()` demonstrates an advantage in runtime over `Ckmeans.1d.dp()` of runtime quadratic in n .

In the second setting (Figure 3), runtime is obtained as a function of number of clusters for `Ckmeans.1d.dp()` and `kmeans()` without constraints on optimality. All input data sets are of the same size 10,000 and were generated from Gaussian mixture models with 2 to 25 components.

In Figure 3, the runtime of `Ckmeans.1d.dp()` increases linearly with the number of clusters k , as does the runtime of `kmeans()`. `kmeans()`, without any constraints on optimality, still shows an advantage in absolute runtime over `Ckmeans.1d.dp()`.

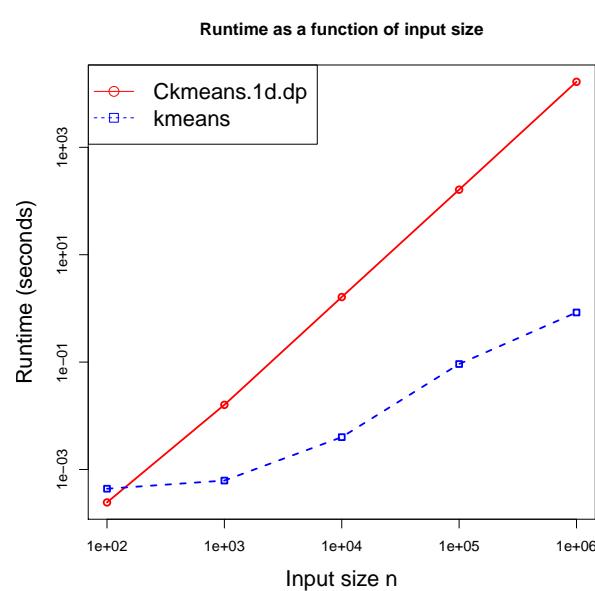


Figure 2: Runtime as a function of input size for `Ckmeans.1d.dp()` and `kmeans()`. `kmeans()` shows an advantage in runtime without any constraints on optimality. The number of clusters is 2 and there is no restart for k -means.

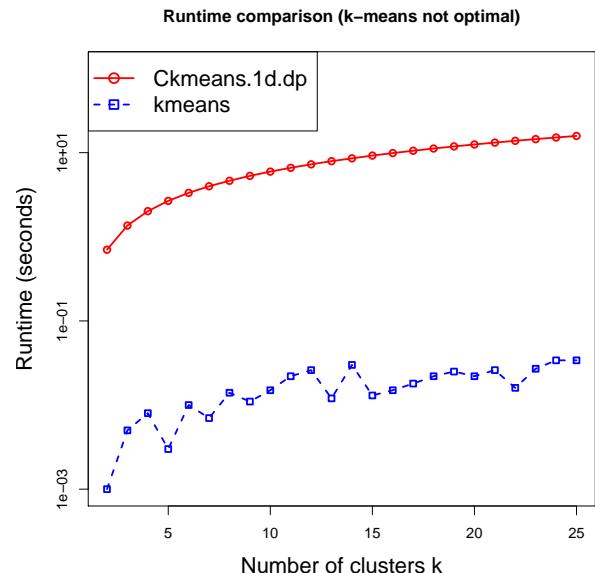


Figure 3: Comparison of runtime as a function of number of clusters between `Ckmeans.1d.dp()` and `kmeans()`. The input data are mixed Gaussian of fixed size 10,000 but with an increasing number of components, representing the clusters in data. Although the k -means algorithm took less runtime, it was run with no restart and thus may not be optimal.

In the third setting (Figure 4), we plot the runtime of `Ckmeans.1d.dp()` and `kmeans()` as a function of the number of clusters, obtained from exactly the same input data in the second setting, but with a constraint on the optimality of k -means such that its `withinss` has a relative difference less than 10^{-6} from the optimal value.

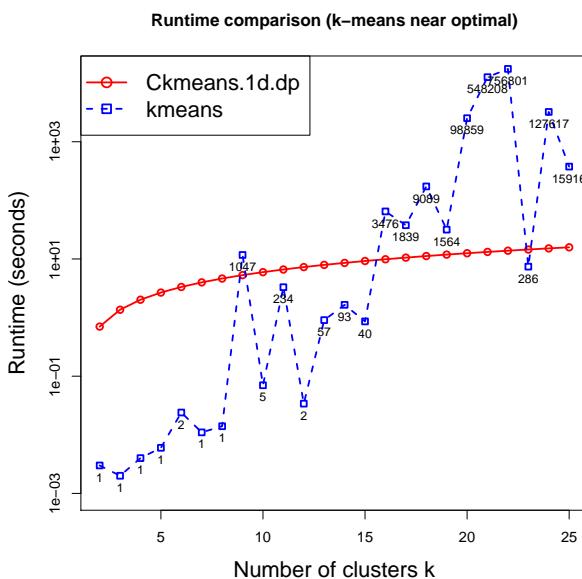


Figure 4: Runtime as a function of number of clusters with restart of `kmeans()` to guarantee a relative difference less than 10^{-6} to the optimal value. The sample size is always 10,000. The number next to each blue square is the number of restarts needed for `kmeans()`. The runtime of `kmeans()` appears to grow exponentially and that of `Ckmeans.1d.dp()` always grows linearly with the number of clusters.

Since `Ckmeans.1d.dp()` returns an optimal result in only one run, its runtime is the same as setting 2. On the other hand, `kmeans()` was restarted a number of times in order to approach an optimal solution within the given tolerance. The number of restarts increases substantially to produce a nearly optimal solution as the number of clusters k increases. As shown in Figure 4, the runtime of `Ckmeans.1d.dp()` increases linearly with k , but the runtime of `kmeans()` appears to increase exponentially with k . The runtime of `kmeans()` almost always far exceeds `Ckmeans.1d.dp()` when k is above 15. This suggests an advantage of `Ckmeans.1d.dp()` over `kmeans()` in both runtime and optimality when k is big.

The R source code for the simulation studies is available from <http://www.cs.nmsu.edu/~joemsong/R-Journal/Ckmeans-Simulation.zip>.

Introduction to the R package `Ckmeans.1d.dp`

The `Ckmeans.1d.dp` package implements the dynamic programming algorithm for 1-D clustering that we described above. In the package, the `Ckmeans.1d.dp()` function performs the clustering on a 1-D input vector using a given number of clusters.

The following example illustrates how to use the package. Figure 5 visualizes the input data and the cluster result obtained by `Ckmeans.1d.dp()` in this example.

```
# a one-dimensional example
# with a two-component Gaussian mixture model
x <- rnorm(50, mean = 1, sd = 0.3)
x <- append(x, rnorm(50, sd = 0.3) )
result <- Ckmeans.1d.dp(x, 2)
plot(x, col = result$cluster)
abline(h = result$centers, col = 1:2, pch = 8,
       cex = 2)
```

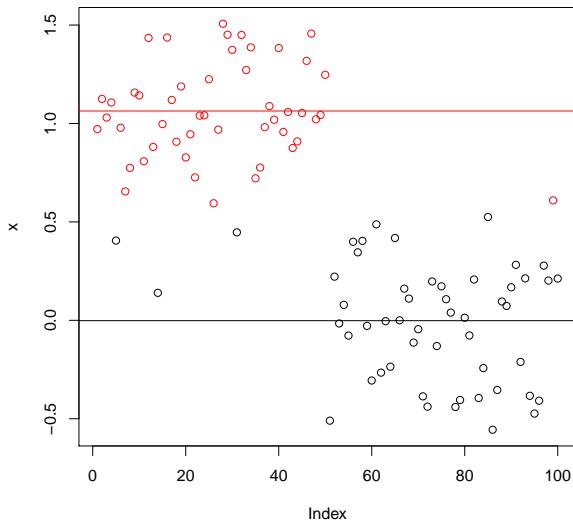


Figure 5: Two clusters obtained from applying function `Ckmeans.1d.dp()` on input x sampled from a Gaussian mixture model with two components. The horizontal axis is the index number of each point in x . The two clusters are indicated using red and black circles. The two horizontal lines represent the centers of each cluster.

Summary

The main purpose of our work is to develop an exact solution to 1-D clustering in a practical amount of time, as an alternative to heuristic k -means algorithms. We thus developed the `Ckmeans.1d.dp` package using dynamic programming to guarantee clustering optimality in $O(n^2k)$ time. The algorithm is implemented in C++ with an interface to the R language. It is provided as a package to R and has been tested under recent operating system versions of Windows, Linux and MacOS. By simulation studies, we demonstrated its advantage over the standard k -means algorithm when optimality is required.

There are two limitations of our `Ckmeans.1d.dp` method. It only applies to 1-D data and its quadratic runtime in the sample size may not be acceptable in all applications.

However, where applicable, the impact of our method is its optimality and repeatability. As our simulation studies show, when the number of clusters is big, our method not only guarantees optimality but also has a fast runtime. In this situation, our method

is preferred. We applied our **Ckmeans.1d.dp** to quantize biological data in the DREAM Challenges (Prill et al., 2010) for qualitative dynamic modeling using generalized logical networks (Song et al., 2009).

It is of great interest if the 1-D dynamic programming strategy can be extended to multiple dimensional spaces so that clustering can be done in polynomial time to k . However, this seems to us to remain open indefinitely.

Acknowledgement

We gratefully acknowledge the financial support to the reported work by the grant number HRD-0420407 from the U.S. National Science Foundation.

Bibliography

- D. Aloise, A. Deshpande, P. Hansen, and P. Popat. NP-hardness of Euclidean sum-of-squares clustering. *Mach. Learn.*, 75(2):245–248, 2009.
- S. Dasgupta and Y. Freund. Random projection trees for vector quantization. *IEEE T. Inform. Theory*, 55(7):3229–3242, 2009.
- J. A. Hartigan and M. A. Wong. Algorithm AS 136: A k-means clustering algorithm. *J. Roy. Stat. Soc. C-App.*, 28(1):100–108, 1979.
- S. A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *J. Theoretical Biology*, 22(3):437–467, 1969.
- S. A. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, USA, June 1993.
- J. Kleinberg and E. Tardos. *Algorithm Design*. Addison-Wesley, USA, 2006.
- S. P. Lloyd. Least squares quantization in PCM. *IEEE T. Inform. Theory*, 28:129 – 137, 1982.
- M. Mahajan, P. Nimborkar, and K. Varadarajan. The planar k-means problem is NP-hard. In *WALCOM '09: Proceedings of the 3rd International Workshop on Algorithms and Computation*, pages 274–285, Berlin, Heidelberg, 2009. Springer-Verlag.
- C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008. ISBN 0521865719, 9780521865715.
- R. J. Prill, D. Marbach, J. Saez-Rodriguez, P. K. Sorger, L. G. Alexopoulos, X. Xue, N. D. Clarke, G. Altan-Bonnet, and G. Stolovitzky. Towards a rigorous assessment of systems biology models: The DREAM3 challenges. *PLoS ONE*, 5(2):e9202, 02 2010.
- M. Song and H. Wang. *Ckmeans.1d.dp: Optimal distance-based clustering for one-dimensional data*, 2011. URL <http://CRAN.R-project.org/package=Ckmeans.1d.dp>. R package version 2.2.
- M. Song, C. K. Lewis, E. R. Lance, E. J. Chesler, R. K. Yordanova, M. A. Langston, K. H. Lodowski, and S. E. Bergeson. Reconstructing generalized logical networks of transcriptional regulation in mouse brain from temporal gene expression data. *EURASIP J. Bioinform. Syst. Biol.*, 2009:1–13, 2009.
- M. Song, R. M. Haralick, and S. Boissinot. Efficient and exact maximum likelihood quantisation of genomic features using dynamic programming. *Int. J. Data Min. Bioinform.*, 4(2):123–141, 2010.

Mingzhou Song
Department of Computer Science
New Mexico State University
United States
joemsong@cs.nmsu.edu

Haizhou Wang
Department of Computer Science
New Mexico State University
United States
hwang@cs.nmsu.edu

Nonparametric Goodness-of-Fit Tests for Discrete Null Distributions

by Taylor B. Arnold and John W. Emerson

Abstract Methodology extending nonparametric goodness-of-fit tests to discrete null distributions has existed for several decades. However, modern statistical software has generally failed to provide this methodology to users. We offer a revision of R's `ks.test()` function and a new `cvm.test()` function that fill this need in the R language for two of the most popular nonparametric goodness-of-fit tests. This paper describes these contributions and provides examples of their usage. Particular attention is given to various numerical issues that arise in their implementation.

Introduction

Goodness-of-fit tests are used to assess whether data are consistent with a hypothesized null distribution. The χ^2 test is the best-known parametric goodness-of-fit test, while the most popular nonparametric tests are the classic test proposed by Kolmogorov and Smirnov followed closely by several variants on Cramér-von Mises tests.

In their most basic forms, these nonparametric goodness-of-fit tests are intended for continuous hypothesized distributions, but they have also been adapted for discrete distributions. Unfortunately, most modern statistical software packages and programming environments have failed to incorporate these discrete versions. As a result, researchers would typically rely upon the χ^2 test or a nonparametric test designed for a continuous null distribution. For smaller sample sizes, in particular, both of these choices can produce misleading inferences.

This paper presents a revision of R's `ks.test()` function and a new `cvm.test()` function to fill this void for researchers and practitioners in the R environment. This work was motivated by the need for such goodness-of-fit testing in a study of Olympic figure skating scoring (Emerson and Arnold, 2011). We first present overviews of the theory and general implementation of the discrete Kolmogorov-Smirnov and Cramér-von Mises tests. We discuss the particular implementation of the tests in R and provide examples. We conclude with a short discussion, including the state of existing continuous and two-sample Cramér-von Mises testing in R.

Kolmogorov-Smirnov test

Overview

The most popular nonparametric goodness-of-fit test is the Kolmogorov-Smirnov test. Given the cumulative distribution function $F_0(x)$ of the hypothesized distribution and the empirical distribution function $F_{data}(x)$ of the observed data, the test statistic is given by

$$D = \sup_x |F_0(x) - F_{data}(x)| \quad (1)$$

When F_0 is continuous, the distribution of D does not depend on the hypothesized distribution, making this a computationally attractive method. Slakter (1965) offers a standard presentation of the test and its performance relative to other algorithms. The test statistic is easily adapted for one-sided tests. For these, the absolute value in (1) is discarded and the tests are based on either the supremum of the remaining difference (the 'greater' testing alternative) or by replacing the supremum with a negative infimum (the 'lesser' hypothesis alternative). Tabulated p -values have been available for these tests since 1933 (Kolmogorov, 1933).

The extension of the Kolmogorov-Smirnov test to non-continuous null distributions is not straightforward. The formula of the test statistic D remains unchanged, but its distribution is much more difficult to obtain; unlike the continuous case, it depends on the null model. Use of the tables associated with continuous hypothesized distributions results in conservative p -values when the null distribution is discontinuous (see Slakter (1965), Goodman (1954), and Massey (1951)). In the early 1970's, Conover (1972) developed the method implemented here for computing exact one-sided p -values in the case of discrete null distributions. The method developed in Gleser (1985) is used to provide exact p -values for two-sided tests.

Implementation

The implementation of the discrete Kolmogorov-Smirnov test involves two steps. First, the particular test statistic is calculated (corresponding to the desired one-sided or two-sided test). Then, the p -value for that particular test statistic may be computed.

The form of the test statistic is the same as in the continuous case; it would seem that no additional work would be required for the implementation, but this is not the case. Consider two non-decreasing functions f and g , where the function f is a step function with jumps on the set $\{x_1, \dots, x_N\}$ and g is continuous (the classical Kolmogorov-Smirnov situation). In

order to determine the supremum of the difference between these two functions, notice that

$$\begin{aligned} & \sup_x |f(x) - g(x)| \\ &= \max_i \left[\max \left(|g(x_i) - f(x_i)|, \right. \right. \\ &\quad \left. \left. \lim_{x \rightarrow x_i} |g(x) - f(x_{i-1})| \right) \right] \end{aligned} \quad (2)$$

$$\begin{aligned} &= \max_i \left[\max \left(|g(x_i) - f(x_i)|, \right. \right. \\ &\quad \left. \left. |g(x_i) - f(x_{i-1})| \right) \right] \end{aligned} \quad (3)$$

Computing the maximum over these $2N$ values (with f equal to $F_{data}(x)$ and g equal to $F_0(x)$ as defined above) is clearly the most efficient way to compute the Kolmogorov-Smirnov test statistic for a continuous null distribution. When the function g is not continuous, however, equality (3) does not hold in general because we cannot replace $\lim_{x \rightarrow x_i} g(x)$ with the value $g(x_i)$.

If it is known that g is a step function, it follows that for some small ϵ ,

$$\begin{aligned} & \sup_x |f(x) - g(x)| = \\ & \max_i (|g(x_i) - f(x_i)|, |g(x_i - \epsilon) - f(x_{i-1})|) \end{aligned} \quad (4)$$

where the discontinuities in g are more than some distance ϵ apart. This, however, requires knowledge that g is a step function as well as of the nature of its support (specifically, the break-points). As a result, we implement the Kolmogorov-Smirnov test statistic for discrete null distributions by requiring the complete specification of the null distribution.

Having obtained the test statistic, the p -value must then be calculated. When an exact p -value is required for smaller sample sizes, the methodology in Conover (1972) is used in for one-sided tests. For two-sided tests, the methods presented in Gleser (1985) lead to exact two-sided p -values. This requires the calculation of rectangular probabilities for uniform order statistics as discussed by Niederhausen (1981). Full details of the calculations are contained in source code of our revised function `ks.test()` and in the papers of Conover and Gleser.

For larger sample sizes (or when requested for smaller sample sizes), the classical Kolmogorov-Smirnov test is used and is known to produce conservative p -values for discrete distributions; the revised `ks.test()` supports estimation of p -values via simulation if desired.

Cramér-von Mises tests

Overview

While the Kolmogorov-Smirnov test may be the most popular of the nonparametric goodness-of-fit tests, Cramér-von Mises tests have been shown to be more powerful against a large class of alternatives hypotheses. The original test was developed by Harald Cramér and Richard von Mises (Cramér, 1928; von Mises, 1928) and further adapted by Anderson and Darling (1952), and Watson (1961). The original test statistic, W^2 , Anderson's A^2 , and Watson's U^2 are:

$$W^2 = n \cdot \int_{-\infty}^{\infty} [F_{data}(x) - F_0(x)]^2 dF_0(x) \quad (5)$$

$$A^2 = n \cdot \int_{-\infty}^{\infty} \frac{[F_{data}(x) - F_0(x)]^2}{F_0(x) - F_0(x)^2} dF_0(x) \quad (6)$$

$$U^2 = n \cdot \int_{-\infty}^{\infty} [F_{data}(x) - F_0(x) - W^2]^2 dF_0(x) \quad (7)$$

As with the original Kolmogorov-Smirnov test statistic, these all have test statistic null distributions which are independent of the hypothesized continuous models. The W^2 statistic was the original test statistic. The A^2 statistic was developed by Anderson in the process of generalizing the test for the two-sample case. Watson's U^2 statistic was developed for distributions which are cyclic (with an ordering to the support but no natural starting point); it is invariant to cyclic reordering of the support. For example, a distribution on the months of the year could be considered cyclic.

It has been shown that these tests can be more powerful than Kolmogorov-Smirnov tests to certain deviations from the hypothesized distribution. They all involve integration over the whole range of data, rather than use of a supremum, so they are best-suited for situations where the true alternative distribution deviates a little over the whole support rather than having large deviations over a small section of the support. Stephens (1974) offers a comprehensive analysis of the relative powers of these tests.

Generalizations of the Cramér-von Mises tests to discrete distributions were developed in Choulakian et al. (1994). As with the Kolmogorov-Smirnov test, the forms of the test statistics are unchanged, and the null distributions of the test statistics are again hypothesis-dependent. Choulakian et al. (1994) does not offer finite-sample results, but rather shows that the asymptotic distributions of the test statistics under the null hypothesis each involve consideration of a weighted sum of independent chi-squared variables (with the weights depending on the particular null distribution).

Implementation

Calculation of the three test statistics is done using the matrix algebra given by Choulakian et al. (1994). The only notable difficulty in the implementation of

the discrete form of the tests involves calculating the percentiles of the weighted sum of chi-squares,

$$Q = \sum_{i=1}^p \lambda_i \chi_{i,1df}^2 \quad (8)$$

where p is the number of elements in the support of the hypothesized distribution. Imhof (1961) provides a method for obtaining the distribution of Q , easily adapted for our case because the chi-squared variables have only one degree of freedom. The exact formula given for the distribution function of Q is given by

$$\mathbb{P}\{Q \geq x\} = \frac{1}{2} + \frac{1}{\pi} \int_0^\infty \frac{\sin[\theta(u, x)]}{u \rho(u)} du \quad (9)$$

for continuous functions $\theta(\cdot, x)$ and $\rho(\cdot)$ depending on the weights λ_i .

There is no analytic solution to the integral in (9), so the integration is accomplished numerically. This seems fine in most situations we considered, but numerical issues appear in the regime of large test statistics x (or, equivalently, small p -values). The function $\theta(\cdot, x)$ is linear in x ; as the test statistic grows the corresponding periodicity of the integrand decreases and the approximation becomes unstable. As an example of this numerical instability, the red plotted in Figure 1 shows the non-monotonicity of the numerical evaluation of equation (9) for a null distribution that is uniform on the set $\{1, 2, 3\}$.

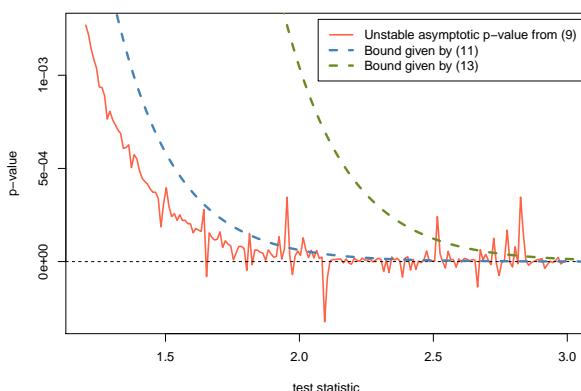


Figure 1: Plot of calculated p -values for given test statistics using numerical integration (red) compared to the conservative chi-squared bound (dashed blue) and the Markov inequality bound (dashed green). The null distribution is uniform on the set $\{1, 2, 3\}$ in this example. The sharp variations in the calculated p -values are a result of numerical instabilities, and the true p -values are bounded by the dashed curves.

We resolve this problem by using a combination of two conservative approximations to avoid the numeri-

cal instability. First, consider the following inequality:

$$\mathbb{P}\left(\sum_{i=1}^p \lambda_i \chi_1^2 \geq x\right) \leq \mathbb{P}\left(\lambda_{\max} \sum_{i=1}^p \chi_1^2 \geq x\right) \quad (10)$$

$$= \mathbb{P}\left(\chi_p^2 \geq \frac{x}{p \lambda_{\max}}\right) \quad (11)$$

The values for the weighted sum can be bounded using a simple transformation and a chi-squared distribution of a higher degree of freedom. Second, consider the Markov inequality:

$$\mathbb{P}\left(\sum_{i=1}^p \lambda_i \chi_1^2 \geq x\right) \leq \mathbb{E}\left[\exp\left(t \sum_{i=1}^p \lambda_i Z_i^2\right)\right] \exp(-tx) \quad (12)$$

$$= \frac{\exp(-tx)}{\sqrt{\prod_{i=1}^p (1 - 2t\lambda_i)}} \quad (13)$$

where the bound can be minimized over $t \in (0, 1/2\lambda_{\max})$. The upper bounds for the p -value given by (11) and (13) are both calculated and the smaller is used in cases where the numerical instability of (9) may be a concern.

The original formulation, numerical integration of (9), is preferable for most p -values, while the upper bound described above is used for smaller p -values (smaller than 0.001, based on our observations of the numerical instability of the original formulation). Figure 1 shows the bounds with the blue and green dashed lines; values in red exceeding the bounds are a result of the numerical instability. Although it would be preferable to determine the use of the bound based on values of the test statistic rather than the p -value, the range of "extreme" values of the test statistic varies with the hypothesized distribution.

Kolmogorov-Smirnov and Cramér-von Mises tests in R

Functions `ks.test()` and `cvm.test()` are provided for convenience in package `dgof`, available on CRAN. Function `ks.test()` offers a revision of R's Kolmogorov-Smirnov function `ks.test()` from recommended package `stats`; `cvm.test()` is a new function for Cramér-von Mises tests.

The revised `ks.test()` function supports one-sample tests for discrete null distributions by allowing the second argument, y , to be an empirical cumulative distribution function (an R function with class "ecdf") or an object of class "stepfun" specifying a discrete distribution. As in the original version of `ks.test()`, the presence of ties in the data (the first argument, x) generates a warning unless y describes a discrete distribution. If the sample size is less than or equal to 30, or when `exact=TRUE`, exact p -values are

provided (a warning is issued when the sample size is greater than 30 due to possible numerical instabilities). When `exact = FALSE` (or when `exact` is unspecified and the sample size is greater than 30) the classical Kolmogorov-Smirnov null distribution of the test statistic is used and resulting *p*-values are known to be conservative though imprecise (see Conover (1972) for details). In such cases, simulated *p*-values may be desired, produced by the `simulate.p.value=TRUE` option.

The function `cvm.test()` is similar in design to `ks.test()`. Its first two arguments specify the data and null distribution; the only extra option, `type`, specifies the variant of the Cramér-von Mises test:

x a numerical vector of data values.

y an `ecdf` or `step-function` (`stepfun`) for specifying the null model

type the variant of the Cramér-von Mises test; `W2` is the default and most common method, `U2` is for cyclical data, and `A2` is the Anderson-Darling alternative.

As with `ks.test()`, `cvm.test()` returns an object of class "htest".

Examples

Consider a toy example with observed data of length 2 (specifically, the values 0 and 1) and a hypothesized null distribution that places equal probability on the values 0 and 1. With the current `ks.test()` function in R (which, admittedly, doesn't claim to handle discrete distributions), the reported *p*-value, 0.5, is clearly incorrect:

```
> stats::ks.test(c(0, 1), ecdf(c(0, 1)))

One-sample Kolmogorov-Smirnov test

data: c(0, 1)
D = 0.5, p-value = 0.5
alternative hypothesis: two-sided
```

Instead, the value of D given in equation (1) should be 0 and the associated *p*-value should be 1. Our revision of `ks.test()` fixes this problem when the user provides a discrete distribution:

```
> library(dgof)
> dgof::ks.test(c(0, 1), ecdf(c(0, 1)))

One-sample Kolmogorov-Smirnov test

data: c(0, 1)
D = 0, p-value = 1
alternative hypothesis: two-sided
```

Next, we simulate a sample of size 25 from the discrete uniform distribution on the integers $\{1, 2, \dots, 10\}$

and show usage of the new `ks.test()` implementation. The first is the default two-sided test, where the exact *p*-value is obtained using the methods of Gleser (1985).

```
> set.seed(1)
> x <- sample(1:10, 25, replace = TRUE)
> x

[1]  3  4  6 10  3  9 10  7  7  1  3  2  7
[14]  4  8  5  8 10  4  8 10  3  7  2  3

> dgof::ks.test(x, ecdf(1:10))

One-sample Kolmogorov-Smirnov test

data: x
D = 0.08, p-value = 0.9354
alternative hypothesis: two-sided
```

Next, we conduct the default one-sided test, where Conover's method provides the exact *p*-value (up to the numerical precision of the implementation):

```
> dgof::ks.test(x, ecdf(1:10),
+   alternative = "g")

One-sample Kolmogorov-Smirnov test

data: x
D^+ = 0.04, p-value = 0.7731
alternative hypothesis:
the CDF of x lies above the null hypothesis
```

In contrast, the option `exact=FALSE` results in the *p*-value obtained by applying the classical Kolmogorov-Smirnov test, resulting in a conservative *p*-value:

```
> dgof::ks.test(x, ecdf(1:10),
+   alternative = "g", exact = FALSE)

One-sample Kolmogorov-Smirnov test

data: x
D^+ = 0.04, p-value = 0.9231
alternative hypothesis:
the CDF of x lies above the null hypothesis
```

The *p*-value may also be estimated via a Monte Carlo simulation:

```
> dgof::ks.test(x, ecdf(1:10),
+   alternative = "g",
+   simulate.p.value = TRUE, B = 10000)

One-sample Kolmogorov-Smirnov test

data: x
D^+ = 0.04, p-value = 0.7717
alternative hypothesis:
the CDF of x lies above the null hypothesis
```

A different toy example shows the dangers of using R's existing `ks.test()` function with discrete data:

```
> dgof::ks.test(rep(1, 3), ecdf(1:3))
```

```
One-sample Kolmogorov-Smirnov test

data: rep(1, 3)
D = 0.6667, p-value = 0.07407
alternative hypothesis: two-sided
```

If, instead, either `exact=FALSE` is used with the new `ks.test()` function, or if the original `stats::ks.test()` is used, the reported *p*-value is 0.1389 even though the test statistic is the same.

We demonstrate the Cramér-von Mises tests with the same simulated data.

```
> cvm.test(x, ecdf(1:10))

Cramer-von Mises - W2

data: x
W2 = 0.057, p-value = 0.8114
alternative hypothesis: Two.sided

> cvm.test(x, ecdf(1:10), type = "A2")

Cramer-von Mises - A2

data: x
A2 = 0.3969, p-value = 0.75
alternative hypothesis: Two.sided
```

We conclude with a toy cyclical example showing that the test is invariant to cyclic reordering of the support.

```
> set.seed(1)
> y <- sample(1:4, 20, replace = TRUE)
> cvm.test(y, ecdf(1:4), type = 'U2')

Cramer-von Mises - U2

data: y
U2 = 0.0094, p-value = 0.945
alternative hypothesis: Two.sided

> z <- y%%4 + 1
> cvm.test(z, ecdf(1:4), type = 'U2')

Cramer-von Mises - U2

data: z
U2 = 0.0094, p-value = 0.945
alternative hypothesis: Two.sided
```

In contrast, the Kolmogorov-Smirnov or the standard Cramér-von Mises tests produce different results after such a reordering. For example, the default Cramér-von Mises test yields *p*-values of 0.8237 and 0.9577 with the original and transformed data `y` and `z`, respectively.

Discussion

This paper presents the implementation of several nonparametric goodness-of-fit tests for discrete null distributions. In some cases the *p*-values are known to

be exact. In others, conservativeness in special cases with small *p*-values has been established. Although we provide for Monte Carlo simulated *p*-values with the new `ks.test()`, no simulations may be necessary necessary for these methods; they were generally developed during an era when extensive simulations may have been prohibitively expensive or time-consuming. However, this does raise the possibility that alternative tests relying upon modern computational abilities could provide even greater power in certain situations, a possible avenue for future work.

In the continuous setting, both of the Kolmogorov-Smirnov and the Cramér-von Mises tests have two-sample analogues. When data are observed from two processes or sampled from two populations, the hypothesis tested is whether they came from the same (unspecified) distribution. With the discrete case, however, the null distribution of the test statistic depends on the underlying probability model, as discussed by Walsh (1963). Such an extension would require the specification of a null distribution, which generally goes unstated in two-sample goodness-of-fit tests. We note that Dufour and Farhat (2001) explored two-sample goodness-of-fit tests for discrete distributions using a permutation test approach.

Further generalizations of goodness-of-fit tests for discrete distributions are described in the extended study of de Wet and Venter (1994). There are existing R packages for certain type of Cramér-von Mises goodness-of-fit tests for continuous distributions. Functions implemented in package **nortest** (Gross, 2006) focus on the composite hypothesis of normality, while package **ADGofTest** (Bellosta, 2009) provides the Anderson-Darling variant of the test for general continuous distributions. Packages **CvM2SL1Test** (Xiao and Cui, 2009a) and **CvM2SL2Test** (Xiao and Cui, 2009b) provide two-sample Cramér-von Mises tests with continuous distributions. Package **cramer** (Franz, 2006) offers a multivariate Cramér test for the two-sample problem. Finally, we note that the discrete goodness-of-fit tests discussed in this paper do not allow the estimation of parameters of the hypothesized null distributions (see Lockhart et al. (2007) for example).

Acknowledgement

We are grateful to Le-Minh Ho for his implementation of the algorithm described in Niederhausen (1981) for calculating rectangular probabilities of uniform order statistics. We also appreciate the feedback from two reviewers of this paper as well as from a reviewer of Emerson and Arnold (2011).

Bibliography

T. W. Anderson and D. A. Darling. Asymptotic theory of certain “goodness of fit” criteria based on

- stochastic processes. *Annals of Mathematical Statistics*, 23:193–212, 1952.
- C. J. G. Bellostas. *ADGofTest: Anderson-Darling GoF test*, 2009. URL <http://CRAN.R-project.org/package=ADGofTest>. R package version 0.1.
- V. Choulakian, R. A. Lockhart, and M. A. Stephens. Cramér-von Mises statistics for discrete distributions. *The Canadian Journal of Statistics*, 22(1):125–137, 1994.
- W. J. Conover. A Kolmogorov goodness-of-fit test for discontinuous distributions. *Journal of the American Statistical Association*, 67(339):591–596, 1972.
- H. Cramér. On the composition of elementary errors. *Skand. Akt.*, 11:141–180, 1928.
- T. de Wet and J. Venter. Asymptotic distributions for quadratic forms with applications to tests of fit. *Annals of Statistics*, 2:380–387, 1994.
- J.-M. Dufour and A. Farhat. Exact nonparametric two-sample homogeneity tests for possibly discrete distributions. Unpublished manuscript, October 2001. URL <http://hdl.handle.net/1866/362>.
- J. W. Emerson and T. B. Arnold. Statistical sleuthing by leveraging human nature: A study of olympic figure skating. *The American Statistician*, 2011. To appear.
- C. Franz. *cramer: Multivariate nonparametric Cramer-Test for the two-sample-problem*, 2006. R package version 0.8-1.
- L. J. Gleser. Exact power of goodness-of-fit tests of kolmogorov type for discontinuous distributions. *Journal of American Statistical Association*, 80(392):954–958, 1985.
- L. A. Goodman. Kolmogorov-Smirnov tests for psychological research. *Psychological Bulletin*, 51:160–168, 1954.
- J. Gross. *nortest: Tests for Normality*, 2006. R package version 1.0.
- J. P. Imhof. Computing the distribution of quadratic forms in normal variables. *Biometrika*, 48:419–426, 1961.
- A. Kolmogorov. Sulla determinazione empirica di una legge di distribuzione. *Giornale dell' Instituto Italiano degli Attuari*, 4:83–91, 1933.
- R. Lockhart, J. Spinelli, and M. Stephens. Cramér-von mises statistics for discrete distributions with unknown parameters. *Canadian Journal of Statistics*, 35(1):125–133, 2007.
- F. Massey. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American Statistical Association*, 46:68–78, 1951.
- H. Niederhausen. Sheffer polynomials for computing exact Kolmogorov-Smirnov and Rényi type distributions. *The Annals of Statistics*, 9(5):923–944, 1981. ISSN 0090-5364.
- M. J. Slakter. A comparison of the Pearson chi-square and Kolmogorov goodness-of-fit tests with respect to validity. *Journal of the American Statistical Association*, 60(311):854–858, 1965.
- M. A. Stephens. Edf statistics for goodness of fit and some comparisons. *Journal of the American Statistical Association*, 69(347):730–737, 1974.
- R. E. von Mises. *Wahrscheinlichkeit, Statistik und Wahrheit*. Julius Springer, Vienna, Austria, 1928.
- J. E. Walsh. Bounded probability properties for Kolmogorov-Smirnov and similar statistics for discrete data. *Annals of the Institute of Statistical Mathematics*, 15:153–158, 1963.
- G. S. Watson. Goodness of fit tests on the circle. *Biometrika*, 48:109–114, 1961.
- Y. Xiao and Y. Cui. *CvM2SL1Test: L1-version of Cramer-von Mises Two Sample Tests*, 2009a. R package version 0.0-2.
- Y. Xiao and Y. Cui. *CvM2SL2Test: Cramer-von Mises Two Sample Tests*, 2009b. R package version 0.0-2.

*Taylor B. Arnold
Yale University
24 Hillhouse Ave.
New Haven, CT 06511 USA
taylor.arnold@yale.edu*

*John W. Emerson
Yale University
24 Hillhouse Ave.
New Haven, CT 06511 USA
john.emerson@yale.edu*

Using the Google Visualisation API with R

by Markus Gesmann and Diego de Castillo

Abstract The **googleVis** package provides an interface between R and the Google Visualisation API to create interactive charts which can be embedded into web pages. The best known of these charts is probably the Motion Chart, popularised by Hans Rosling in his TED talks. With the **googleVis** package users can easily create web pages with interactive charts based on R data frames and display them either via the local R HTTP help server or within their own sites.

Motivation

In 2006 Hans Rosling gave an inspiring talk at TED¹ about social and economic developments in the world over the past 50 years, which challenged the views and perceptions of many listeners. Rosling had used extensive data analysis to reach his conclusions. To visualise his talk, he and his colleagues at Gapminder had developed animated bubble charts, see Figure 1.

Rosling's presentation popularised the idea and use of interactive charts, and as a result the software behind Gapminder was bought by Google and integrated as motion charts into their Visualisation API² one year later.

In 2010 Sebastián Pérez Saaibi (Saaibi, 2010) presented at the R/Rmetrics Workshop on Computational Finance and Financial Engineering the idea to use Google motion charts to visualise R output with the **rsp** package (Bengtsson, 2011).

Inspired by those talks and the desire to use interactive data visualisation tools to foster the dialogue between data analysts and others, the authors of this article started the development of the **googleVis** package (Gesmann and de Castillo, 2011).

Google Visualisation API

The Google Visualisation API allows users to create interactive charts as part of Google documents, spreadsheets and web pages. This text will focus on the usage of the API as part of web sites.

The Google Public Data Explorer (<http://www.google.com/publicdata/home>) provides a good example, demonstrating the use of interactive charts and how they can help to analyse data. The charting data can either be embedded into the html file or read dynamically. The key to the Google Visualisation API

is that the data is structured in a “DataTable”, and this is where the **googleVis** package helps. It uses the functionality of the **rjsonio** package (Temple Lang, 2011) to transform R data frames into JSON³ objects as the basis for a DataTable.

As an example, we will look at the html code of a motion chart from Google's visualisation gallery, which generates output similar to Figure 1:

```

1 <html>
2 <head>
3   <script type="text/javascript"
4     src="http://www.google.com/jsapi">
5   </script>
6   <script type="text/javascript">
7     google.load('visualization', '1',
8       {'packages':['motionchart']});
9   google.setOnLoadCallback(drawChart);
10  function drawChart() {
11    var data=new google.visualization.DataTable();
12    data.addColumn('string', 'Fruit');
13    data.addColumn('date', 'Date');
14    data.addColumn('number', 'Sales');
15    data.addColumn('number', 'Expenses');
16    data.addColumn('string', 'Location');
17    data.addRows([
18      ['Apples',new Date(1988,0,1),1000,300,'East'],
19      ['Oranges',new Date(1988,0,1),1150,200,'West'],
20      ['Bananas',new Date(1988,0,1),300,250,'West'],
21      ['Apples',new Date(1989,6,1),1200,400,'East'],
22      ['Oranges',new Date(1989,6,1),750,150,'West'],
23      ['Bananas',new Date(1989,6,1),788,617,'West']
24    ]);
25    var chart=new google.visualization.MotionChart(
26      document.getElementById('chart_div'));
27    chart.draw(data, {width: 600, height:300});
28  }
29 </script>
30 </head>
31 <body>
32 <div id="chart_div"
33   style="width:600px; height:300px;">
34 </div>
35 </body>
36 </html>
```

The code and data are processed and rendered in the browser and is not submitted to any server⁴.

You will notice that the above html code has five generic parts⁵:

- references to Google's AJAX (l. 4) and Visualisation API (ll. 7–8),
- data to visualise as a DataTable (ll. 11–24),
- an instance call to create the chart (ll. 25–26),

¹http://www.ted.com/talks/hans_rosling_shows_the_best_stats_you_ve_ever_seen.html

²<http://code.google.com/apis/visualization/documentation/index.html>

³<http://www.json.org/>

⁴http://code.google.com/apis/visualization/documentation/gallery/motionchart.html#Data_Policy

⁵For more details see http://code.google.com/apis/chart/interactive/docs/adding_charts.html

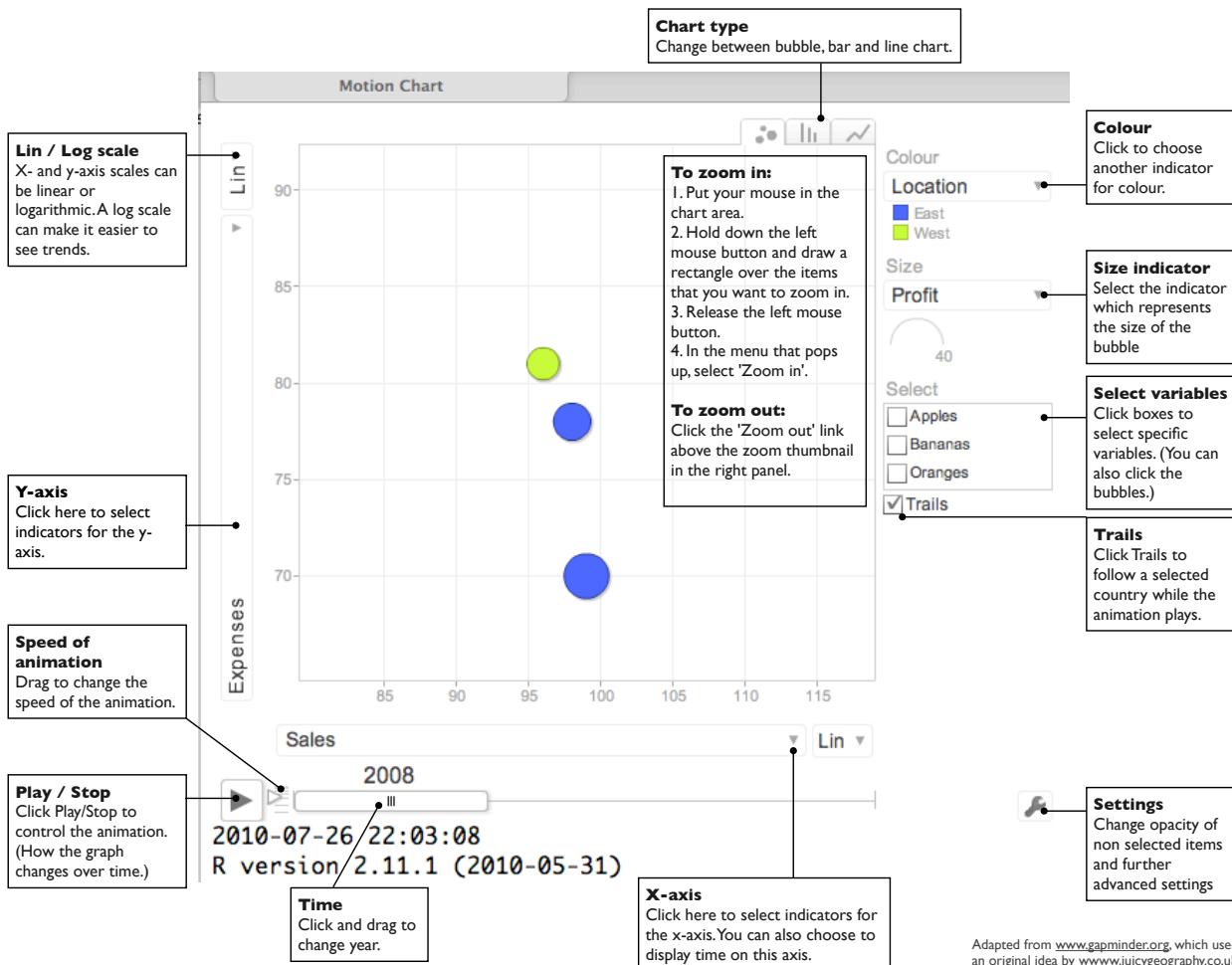


Figure 1: Overview of a Google Motion Chart. Screenshot of the output of `plot(gvisMotionChart(Fruits, idvar = 'Fruit', timevar = 'Year'))`

- a method call to draw the chart including options, shown here as width and height (l. 27),
- an HTML `<div>` element to add the chart to the page (ll. 32–34).

These principles hold true for most of the interactive charts of the Google Visualisation API.

However, before you use the API you should read the [Google Visualisation API Terms of Service](#)⁶ and the [Google Maps/Google Earth APIs Terms of Service](#)⁷.

The `googleVis` package

The `googleVis` package provides an interface between R and the Google Visualisation API. The functions of the package allow the user to visualise data stored in R data frames with the Google Visualisation API.

Version (0.2.12) of the package provides interfaces to Motion Charts, Annotated Time Lines, Geo Maps,

Maps, Geo Charts, Intensity Maps, Tables, Gauges, and Tree Maps, as well as Line-, Bar-, Column-, Area-, Combo-, Scatter-, Candlestick-, Pie- and Org Charts; see Figure 2 for some examples.

The output of a `googleVis` function is html code that contains the data and references to JavaScript functions hosted by Google. A browser with an Internet connection is required to view the output, and for Motion Charts, Geo Maps and Annotated Time Lines also Flash. The actual chart is rendered in the browser.

Please note that Flash charts may not work when loaded as a local file due to security settings, and therefore may require to be displayed via a web server. Fortunately, R comes with an internal HTTP server which allows the `googleVis` package to display pages locally. Other options are to use the `R.rsp` package or `RApache` (Horner, 2011) with `brew` (Horner, 2011). Both `R.rsp` and `brew` have the capability to extract and execute R code from html code, similar to the approach taken by `Sweave` (Leisch, 2002) for `LATEX`.

⁶<http://code.google.com/apis/visualization/terms.html>

⁷<http://code.google.com/apis/maps/terms.html>

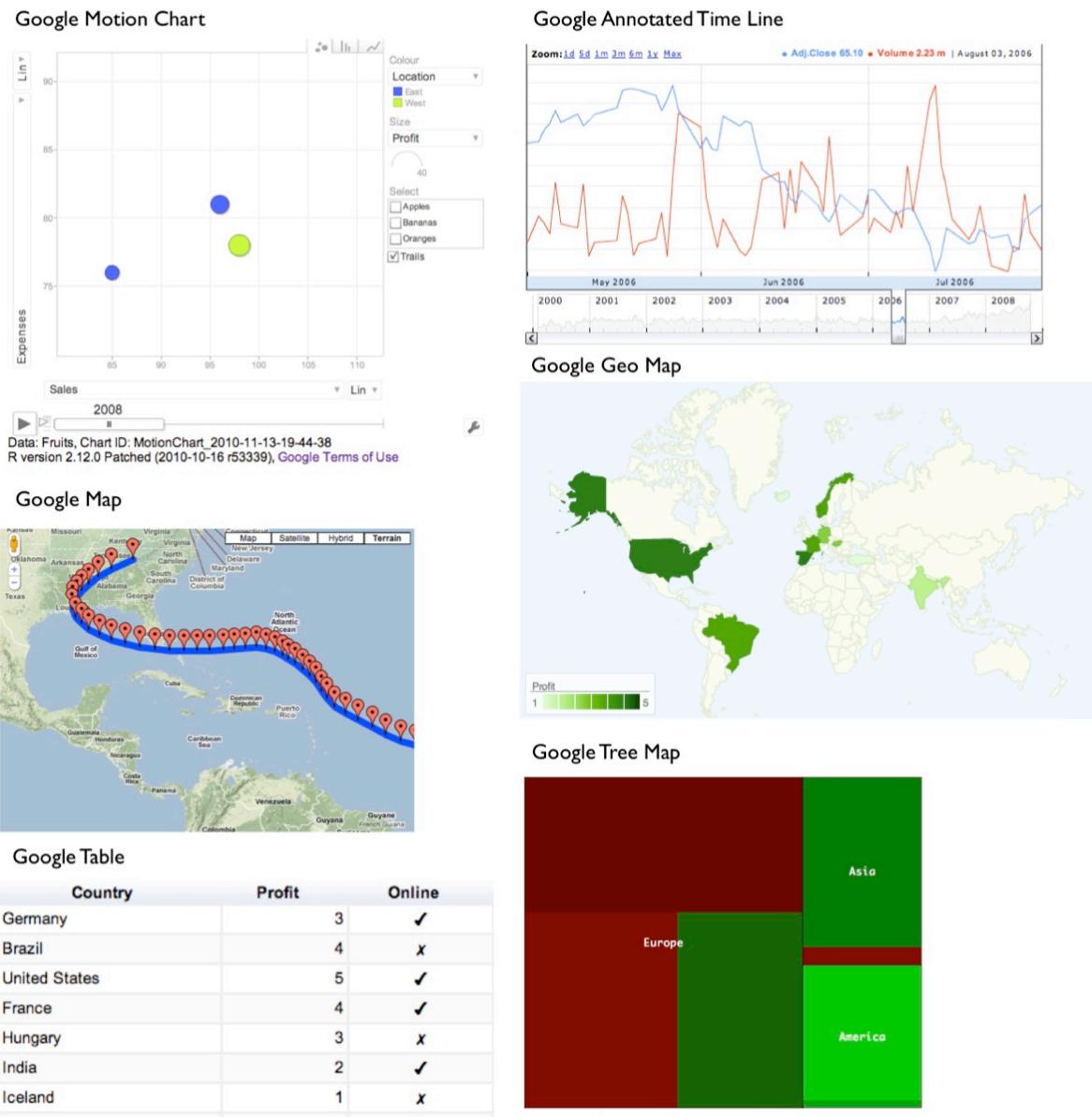


Figure 2: Screenshot of some of the outputs of `demo(googleVis)`. Clockwise from top left: `gvisMotionChart`, `gvisAnnotatedTimeLine`, `gvisGeoMap`, `gvisTreeMap`, and `gvisTable`.

The individual functions of the **googleVis** package are documented in detail in the help pages and package vignette. Here we will cover only the principles of the package.

As an example we will show how to generate a Motion Chart as displayed in Figure 1. It works similarly for the other APIs. Further examples are covered in the demos of the **googleVis** package and on the project Google Code site.

The design of the visualisation functions is fairly generic. The name of the visualisation function is '`gvis`' followed by the chart type. Thus for the Motion Chart we have:

```
gvisMotionChart(data, idvar = 'id',
                 timevar = 'date', options = list())
```

Here `data` is the input `data.frame` and the arguments `idvar` and `timevar` specify the column names of the `id` variable and time variable for the plot, while display options are set in an optional list. The options and data requirements follow those of the Google Visualisation API and are documented in the help pages, see `help('gvisMotionChart')`.

The output of a **googleVis** function is a list of lists (a nested list) containing information about the chart type, chart id, and the html code in a sub-list split into header, chart, caption and footer.

The idea behind this concept is that users get a complete web page while at the same time they can extract only specific parts, such as the chart. This is particularly helpful if the package functions are used in solutions where the user wants to feed the visuali-

sation output into other sites, or would like to embed them into rsp-pages, or use RApache or Google Gadgets.

The output of a **googleVis** function will be of class "gvis" and "list". Generic print (`print.gvis`) and plot (`plot.gvis`) methods exist to ease the handling of such objects.

To illustrate the concept we shall create a motion chart using the `Fruits` data set.

Motion chart example

Following the documentation of the Google Motion Chart API we need a data set which has at least four columns: one identifying the variable we would like to plot, one time variable, and at least two numerical variables; further numerical and character columns are allowed.

As an example we use the `Fruits` data set:

```
R> data(Fruits)
R> Fruits[, -7] # ignore column 7

  Fruit Year Location Sales Expenses Profit
1 Apples 2008    West     98      78     20
2 Apples 2009    West    111      79     32
3 Apples 2010    West     89      76     13
4 Oranges 2008   East     96      81     15
5 Bananas 2008   East     85      76      9
6 Oranges 2009   East     93      80     13
7 Bananas 2009   East     94      78     16
8 Oranges 2010   East     98      91      7
9 Bananas 2010   East     81      71     10
```

Here we will use the columns 'Fruit' and 'Year' as id and time variable respectively.

```
R> M <- gvisMotionChart(Fruits, idvar = "Fruit",
                         timevar = "Year")
```

The structural output of `gvisMotionChart` is a list of lists as described above:

```
R> str(M)

List of 3
$ type   : chr "MotionChart"
$ chartid: chr "MotionChartID12ae2fff"
$ html    :List of 4
..$ header : chr "<!DOCTYPE html PUBLIC ..."
..$ chart   : Named chr [1:7] "<!-- Moti ..."
... .- attr(*, "names")= chr [1:7] "jsH ...
..$ caption: chr "<div><span>Data: Fruit...
..$ footer  : chr "\n<!-- htmlFooter -->\...
- attr(*, "class")= chr [1:2] "gvis" "list"
```

The first two items of the list contain information about the chart type used and the individual chart id. The html output is a list with header, chart, caption and footer. This allows the user to extract only certain parts of the page, or to create a complete html page.

The header part of the html page has only basic html and formatting tags and provides a simple layout, see

```
R> print(M, 'header') # output not shown here
```

The actual Google visualisation code is stored with the data in the named character vector `chart` of the html list, see

```
R> print(M, 'chart') # output not shown here
```

The sub-items of the chart object give the user more granular access to JavaScript functions and html tags of the visualisation. A basic chart caption and html footer complete the html list:

```
R> print(M, 'caption') # output not shown here
R> print(M, 'footer') # output not shown here
```

Displaying "gvis" objects

To display the page locally, type:

```
R> plot(M) # returns invisibly the file name
```

The `plot` method for "gvis" objects creates html files in a temporary folder using the type and chart id information of the object and it will display the output using the R HTTP help web server locally. The R command `tempdir()` shows the path of the per-session temporary directory.

Further examples are part of the **googleVis** demos, including one showing how a Geo Map can be animated with additional JavaScript, see `demo(package = "googleVis")`.

Combining charts with `gvisMerge`

The function `gvisMerge` takes two "gvis" objects and merges the underlying components into one page. The charts are aligned either horizontally or vertically next to each other in an HTML table.

The output of `gvisMerge` is a "gvis" object again. This allows us to apply the same function iteratively to create more complex chart layouts. The following example, see Figure 3, aligns a Geo Chart and Table below each other, and combines the output with a Motion Chart to the right.

```
G <- gvisGeoChart(Exports, "Country", "Profit",
                  options = list(width = 200, height = 100))
T <- gvisTable(Exports,
              options = list(width = 200, height = 270))
M <- gvisMotionChart(Fruits, "Fruit", "Year",
                      options = list(width = 400, height = 370))
GT <- gvisMerge(G, T, horizontal = FALSE)
GTM <- gvisMerge(GT, M, horizontal = TRUE,
                 tableOptions =
                   "bgcolor = \"#CCCCCC\" cellspacing = 10")
plot(GTM)
```

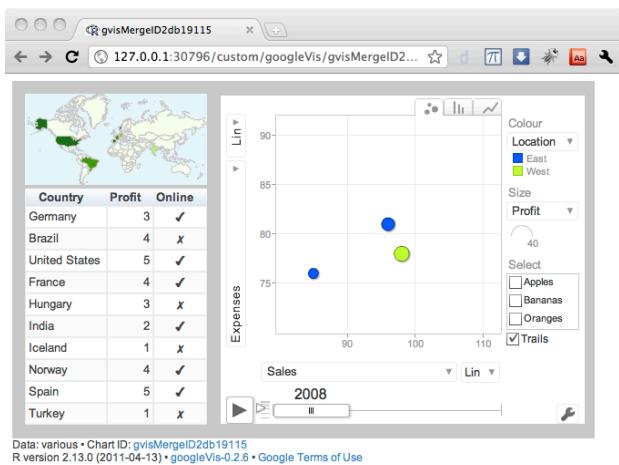


Figure 3: Three charts combined with `gvisMerge`.

Embedding googleVis in web sites dynamically

With the R packages `R.rsp` and `brew` we have two options to integrate R snippets into html code. While the `R.rsp` package comes with its own internal web server, `brew` requires the Apache HTTP server with the RAapache module installed.

Both packages allow the user to embed R code into html code. The R code is filtered by the `R.rsp` or RAapache web server and executed at run time. As an example, we can insert the above motion chart into a page:

```
<html>
<body>
<% library(googleVis) %>
<% M <- gvisMotionChart(Fruits, idvar="Fruit",
  timevar="Year") %>
<%= M$html$chart %>
</body>
</html>
```

The syntax of `R.rsp` and `brew` are very similar. The R code included in `<% ... %>` is executed when read by the HTTP server, but no R output will be displayed. To insert the R output into the html code we have to add an equal sign, `<%= ... %>`, which acts as a `cat` statement. In the example above the chart code is embedded into the html code.

Examples for both approaches are part of the `googleVis` package. For more information see the vignettes and help files of the packages.

Summary

Combining R with the Google Visualisation API enables users to quickly create powerful analysis tools,

which can be shared online. The user interface is easily accessible both to data and non-data analysts.

The “Statistics Relating to Lloyd’s” site (<http://www.lloyds.com/stats>) is an example where the functions of the `googleVis` package were used to create a data visualisation page. Further case studies and links to alternative packages are available on the project Google Code site⁸.

Bibliography

- H. Bengtsson. `R.rsp`: R server pages. <http://CRAN.R-project.org/package=R.rsp>, 2011. R package version 0.7.0.
 - M. Gesmann and D. de Castillo. `googleVis`: Using the Google Visualisation API with R. <http://code.google.com/p/google-motion-charts-with-r/>, 2011. R package version 0.2.12.
 - Jeffrey Horner. `brew`: Templating framework for report generation. <http://CRAN.R-project.org/package=brew>, 2011. R package version 1.0-6.
 - Jeffrey Horner. RAapache: Web application development with R and Apache. <http://www.rapache.net/>, 2011.
 - F. Leisch. Sweave: Dynamic generation of statistical reports using literate data analysis. In W. Härdle and B. Rönz, editors, *Compstat 2002 — Proceedings in Computational Statistics*, pages 575–580. Physica Verlag, Heidelberg, 2002. ISBN 3-7908-1517-9.
 - S. P. Saaibi. R/RMETRICS Generator Tool for Google Motion Charts. 4th R/Rmetrics User and Developer Workshop, Meielisalp, Lake Thune, Switzerland, June 27 - July 1, 2010. <https://www.rmetrics.org/>.
 - D. Temple Lang. RJSONIO: Serialize R objects to JSON, JavaScript Object Notation. <http://CRAN.R-project.org/package=RJSONIO>, 2011. R package version 0.96-0.
- Markus Gesmann
googleVis project
London
UK
markus.gesmann@gmail.com*
- Diego de Castillo
googleVis project
Cologne
Germany
decastillo@gmail.com*

⁸<http://code.google.com/p/google-motion-charts-with-r/>

GrapheR: a Multiplatform GUI for Drawing Customizable Graphs in R

by Maxime Hervé

Abstract This article presents **GrapheR**, a Graphical User Interface allowing the user to draw customizable and high-quality graphs without knowing any R commands. Six kinds of graph are available: histograms, box-and-whisker plots, bar plots, pie charts, curves and scatter plots. The complete process is described with the examples of a bar plot and a scatter plot illustrating the legendary puzzle of African and European swallows' migrations.

Observation and aims

Unlike statistical software based on a Graphical User Interface (*GUI*), such as Minitab® or Stata®, the R language allows you to control exactly how your data will be displayed. This comes at a cost: you must know the exact arguments of the commands to get the result you want.

In this context, beginners find that drawing elaborate graphs is often a long and difficult process made up of lots of trials and slight code modifications. Some well-known R GUIs already exist (e.g. **R Commander** (Fox, 2005), **JGR** (Helbig et al., 2005), **Sci-Views R** (Grosjean, 2010), **Rattle** (Williams, 2009) or **playwith** (Andrews, 2010)), but the graphs they produce have mainly an explorative function in data analysis, and are not customizable to get publication-ready material.

Therefore, many R users, especially beginners (but not only), go back to Excel®-like software and their clickable interfaces to quickly draw the graphs they want to publish. In the compromise between speed and simplicity vs. graph quality, they must sacrifice graph quality in the process. This is regrettable, because R can do lots of things Excel® cannot, by combining high and low-level graphical functions. The goal of the **GrapheR** (Hervé, 2011) package is therefore to combine the simplicity of a GUI with the powerful capabilities and the graphical quality of R.

To be immediately accessible to beginners, **GrapheR** does not require any knowledge of the R language. Indeed, the loading of the dataset, the choice of the variables to be represented and the configuration of all graphical options are made with menus, checkboxes and other clickable tools.

The visual structure and functioning of the GUI are entirely based on the **tcltk** package developed by Peter Dalgaard (Dalgaard, 2001, 2002), which adapts the Tcl/Tk language to R.

Launching the interface

As with any other package, **GrapheR** is loaded via `library(GrapheR)`, `require(GrapheR)` or the 'Packages' menu of the Windows R GUI.

Launching the interface is the only step that requires the user to enter a command: `run.GrapheR()`. The interface opens and the console can now be reduced.

Description of the interface

The interface is divided into three blocks (Figure 1):

1. The navigation bar: it contains seven groups of buttons, each corresponding to one (obligatory or optional) step of the process. From left to right:
 - A. Loading and modifying the dataset.
 - B. Setting up a graph. From left to right: histogram, box-and-whisker plot, bar plot, pie chart, curve and scatter plot.
 - C. Opening a new graphics device: when a graph is drawn, it is in the active window or in a new device if none is open. This button allows the user to open a new graphics device in which several graphs can be drawn.
 - D. Draw a graph.
 - E. Adding elements to the graph. From left to right: add a horizontal line, add a vertical line, add any other kind of line, add a theoretical distribution curve, add text, and add *p*-values.
 - F. Saving graph(s).
 - G. Divers options: user language and help.
2. The messages frame: when the mouse is over some specific elements of the GUI or when some specific actions are performed, messages are displayed in this frame. Three kinds of message can be displayed:
 - **in blue**: informative messages,
 - **in green**: warnings, for when particular attention should be paid to a specific point (for example: if this option is chosen, then this one can not be),
 - **in red**: error messages, for when the action requested can not be completed. The message indicates the origin of the error.

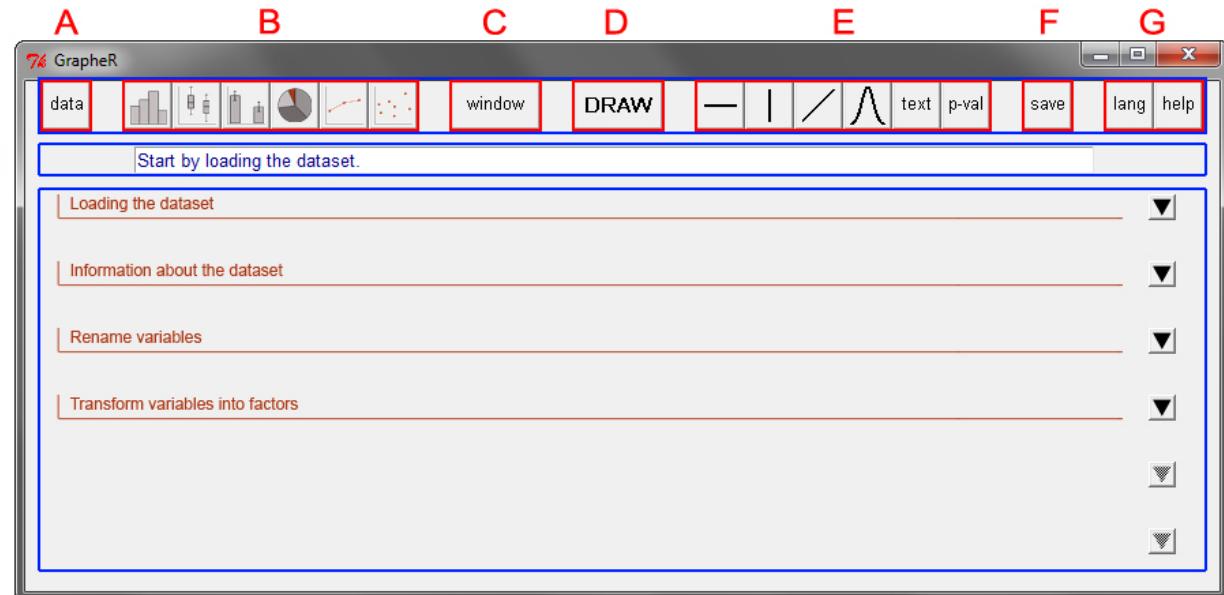


Figure 1: User interface – global view (display under Windows 7)

3. The settings block: it is divided into four to six sub-blocks, each containing options relating to a given theme: general parameters of the graph, title of the graph, legend... Each sub-block can be opened or closed with the corresponding arrow situated on the right of the interface. Of course, defined settings are not lost when a sub-block is closed.

- by converting variables into factors. The conversion can be applied to variables of class "character" or to numeric variables (in this case values can be grouped into classes). The latter case is necessary when a factor is numerically coded, e.g. a binary factor (0/1), otherwise R would treat it as a numeric variable.

Use

The examples shown here are based on the dataset provided in the package, called 'Swallows'. To load it, use the command `data(Swallows)`. This (fictional!) dataset exemplifies the legendary puzzle of African and European swallows' migrations (Gilliam and Jones, 1975).

Loading and modifying the dataset

The first sub-block (Figure 2) allows you to load the dataset. Data can be imported from an external file ('txt' and 'csv' extensions are available so far) or can be an already existing R object of class "data.frame" (i.e. a table).

The next sub-block allows you to get information on the dataset structure. When a variable is selected in the list on the left, its type (numeric, factor, logical...) and its summary are displayed in the frame on the right.

The next two sub-blocks allow the dataset to be modified if needed:

- by renaming variables (for example if the dataset does not contain their names),

Setting up a graph

Once the dataset is ready, click on the button corresponding to the type of graph to be drawn (histogram, box-and-whisker plot, bar plot, pie chart, curve or scatter plot).

Whatever the type of graph chosen, all parameters have a default value except general parameters – which correspond to variable(s) to be represented. Hence, to quickly draw a graph, only general parameters must be defined. If they are not (or not all) defined, an error occurs when trying to draw the graph.

Here the aim is to draw two graphs: a barplot displaying a size comparison of African and European swallows, and a scatter plot displaying the relationship between weight and size in the two species. Let's begin with the barplot. We start by clicking on the 'Bar plot' button in the navigation bar.

We want to show mean sizes of swallows depending on their species. In the general parameters, we hence choose the 'Means' type (bar plots can also display proportions), 'Size' as the variable to be represented and 'Species' as the obligatory factor (Figure 3). A second, optional, factor could be added, if for example we would like to display mean sizes of swallows depending on their species and their sex (you can try to do it, the dataset contains a 'Sex' factor).

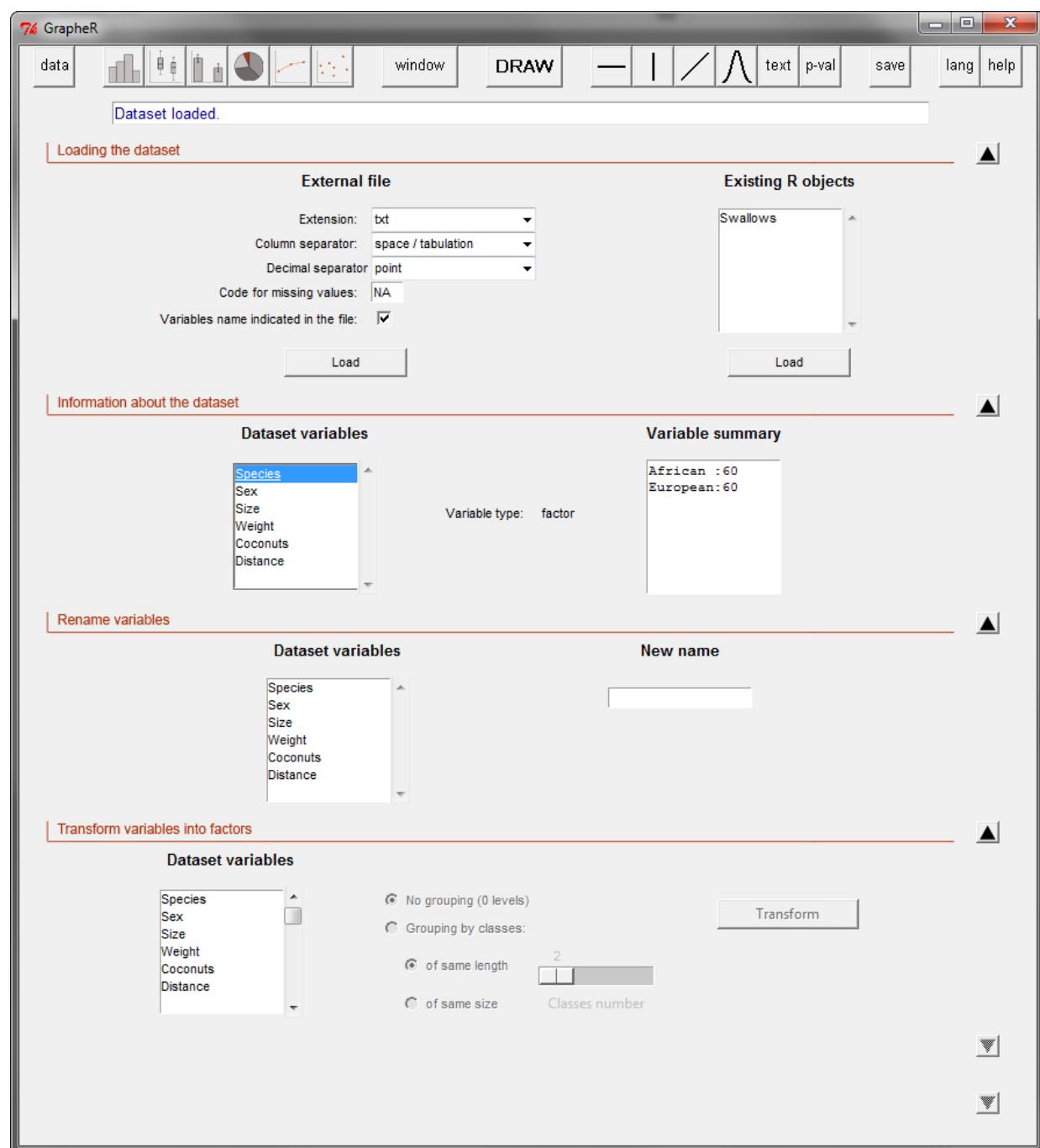


Figure 2: Loading the dataset

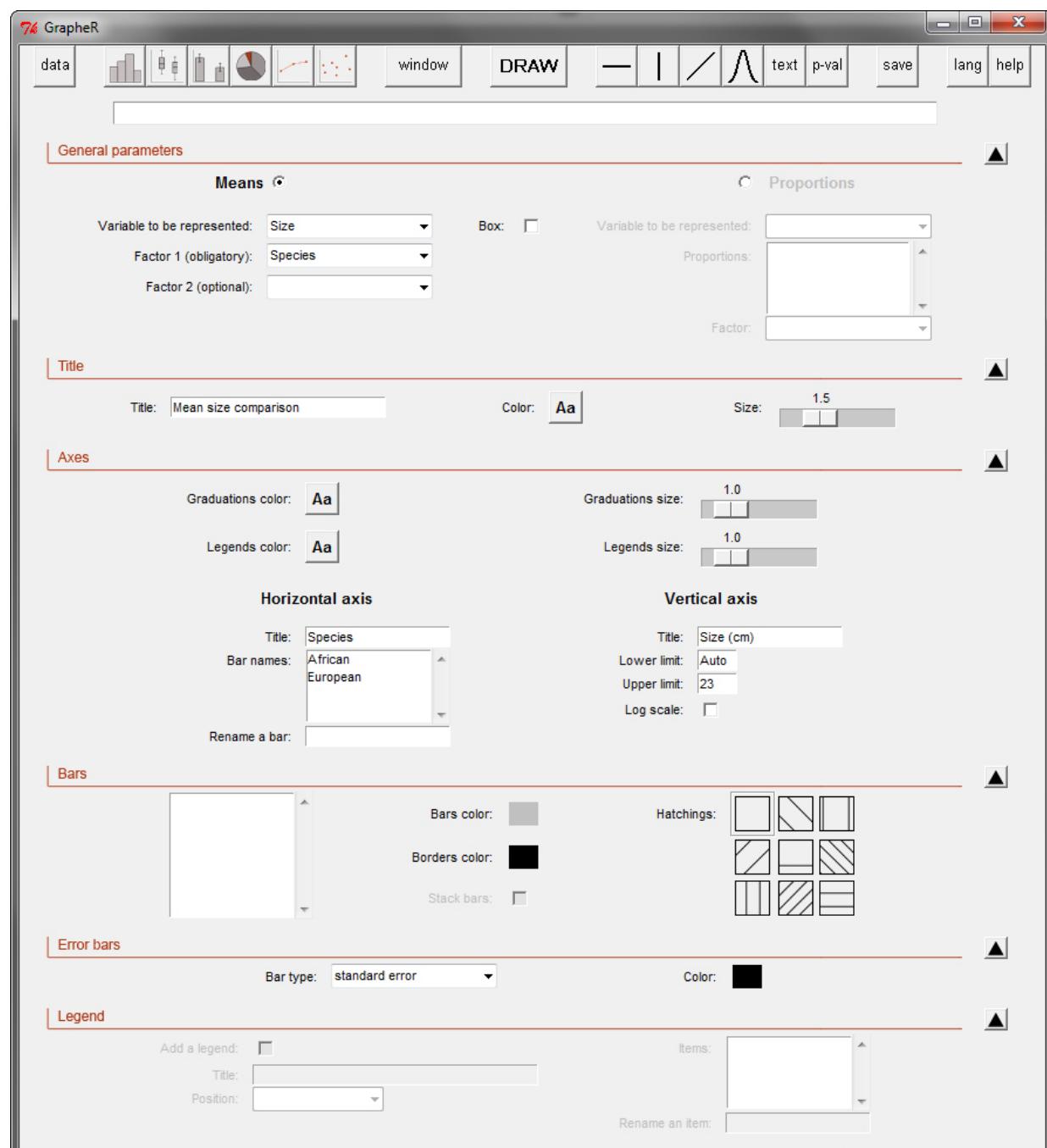


Figure 3: Illustrative bar plot

Graph title and axis legends are optional (no text is written by default). In our example we call our graph 'Mean size comparison' and name the axes 'Species' (horizontal axis) and 'Size (cm)' (vertical axis). Bar names are by default the names of the corresponding levels of factor 1. However we could change this, which would only change the names *displayed* on the graph (and not the ones in the dataset). This allows you to use names containing several words, spaces or accents on your graph (which is either impossible or strongly discouraged for object names).

Lower and upper limits of the vertical axis can be changed, and a logarithmic scale can be used. If limit values are left on 'Auto', **GrapheR** will adjust them automatically. Here the lower limit is left on 'Auto', whilst the upper limit is set to '23', for reasons explained later.

If no second factor is defined, all bars are by default solid white with a black border and no hatchings. We can set these three parameters (bar color, border color and hatchings), but in this example we simply change bar color to a classic grey. If a second factor is defined, one color/hatching pattern can be attributed by level of this factor. In this case colors are by default set to different shades of grey (borders are always black). Bars can be stacked in this situation, but if you choose this option, no error bar can be drawn.

Means and percentages in scientific bar plots are nearly always supplied with error bars, but getting those bars right using the command line is notoriously challenging for R beginners. **GrapheR** makes it easy to draw error bars to represent either the standard deviation, the standard error of the mean or the 95% confidence interval of the mean. Here we choose for example 'standard errors'.

When a second factor is defined, a legend box can be added to the graph. Its items are by default the levels of this factor. As for bar names, legend items can be renamed, which will not change actual level names in the dataset. A title can also be added to the legend. Finally, its position on the graph can be set.

Once all options have been thus configured, just click on the 'DRAW' button of the navigation bar... *et voilà!* By default the graph is produced in the active graphics device, but here we want to display two graphs in the same window. Hence we click on the 'window' button of the navigation bar before drawing our bar plot.

Opening a new graphics device

A dialog box opens on the right of the interface, allowing specification of the number of graphs to be drawn in the device to be created, and the background color of this device (Figure 4).

It is possible to draw up to 16 graphs in the same device, shared between four rows and four columns. However, the larger the number of graphs to be drawn, the smaller the space allocated to each.

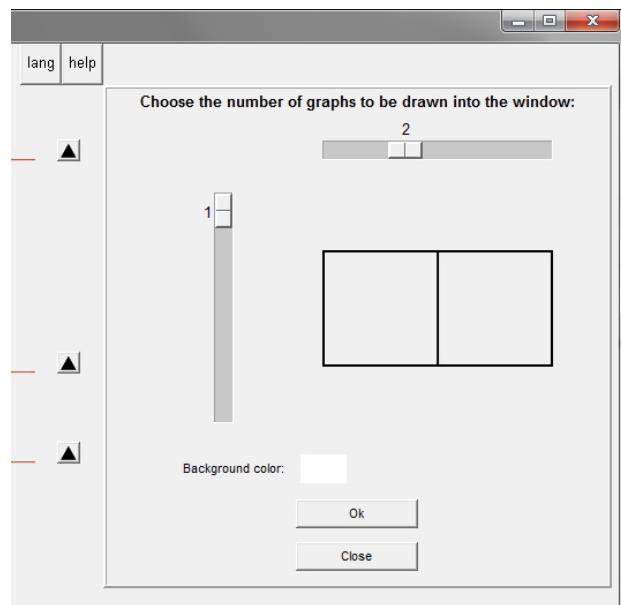


Figure 4: Opening a new graphics device

After creating the new graphics device, we can draw our graph by clicking on the 'DRAW' button.

Adding elements to the graph

Once the graph is drawn, elements can be added to complete it:

- one or several horizontal line(s)
- one or several vertical line(s)
- any other kind of line(s)
- one or several theoretical distribution curve(s): only on density histograms
- text
- *p*-values: only on bar plots

These elements are always added to the last graph drawn.

For each element, clicking on the corresponding button in the navigation bar opens a dialog box on the right of the interface.

In our bar plot example, let's assume that we had used R beforehand to perform a statistical test to compare the sizes of our two species. We can now use the *p*-values tool to add the result of this test on the graph: just click on the 'p-val' button.

Two designs are available, depending on the number of bars to be compared (Figure 5). Whatever the design, the process is the same: enter the text you want, customize its size and color if needed, and finally click on the 'Select' button. Then just click on the bars to be compared on the graph.

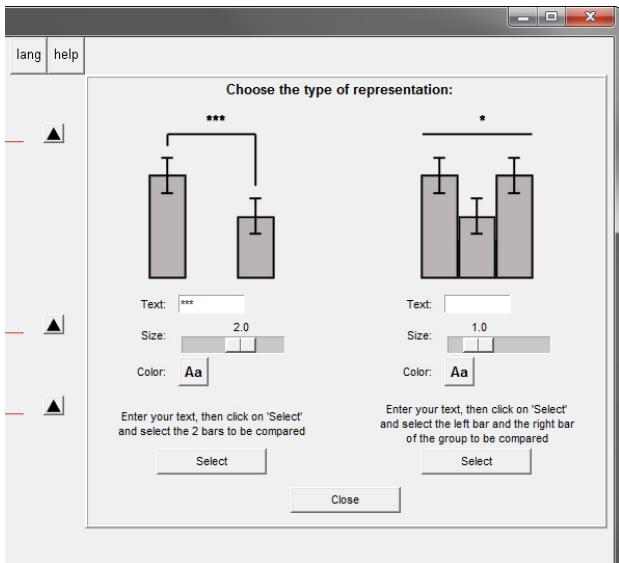


Figure 5: Adding p -values to the bar plot

The result, sometimes, will not look good the first time. It is because there was not enough space above the bars to add text. If you want to add text above your graph, remember to allow some space by setting a higher upper limit for the vertical axis (this is the origin of the value '23' in Figure 3). If you omitted this, no problem, drawing the graph again will be a matter of seconds in **Grapher**. The first graph is now ready (left part of Figure 6).

Second graph

The second graph we want to draw is a scatter plot displaying the relationship between weight and size in the two species. We start by clicking on the 'Scatter plot' button of the navigation bar.

In the general parameters, we define 'Weight' as the X variable and 'Size' as the Y variable (Figure 7).

To draw a scatter plot by species, we add an optional factor, 'Species', and select its two levels from the list. Finally, for aesthetic reasons we choose to draw a box around the graph by ticking the 'Box' checkbox.

We add a title for the graph ('Weight - size relationship'), for the horizontal axis ('Weight (g)') and for the vertical axis ('Size (cm)').

For each scatter plot, a different point symbol (as well as its color and size) can be defined. By default all symbols are empty circles. When only one scatter plot is drawn the default color is black, whereas when several are drawn colors are set to different shades of grey. Here we choose full circles for the two scatter plots and change only the second color (corresponding to the 'European' level) to a dark red.

A line/curve can be added by scatter plot, chosen among different types:

- a linear regression line (performed with the least squares method)
- a linear regression line (performed with the least rectangles method)
- a quadratic regression line
- a simple tendency curve

Here no variable is independent or dependent, but both are interdependent. We hence choose to add a linear regression line performed with the least rectangles method to each scatter plot. Among the three available line types (full, large dashed or fine dashed), we select 'full'.

We add a legend box by ticking the 'Add a legend' checkbox. We give it a title ('Species') and set its position to 'top - left'. We leave the default items, which correspond to levels of the factor defined in the general parameters.

Finally, we click on the 'DRAW' button of the navigation bar... the second graph is born (Figure 6)!

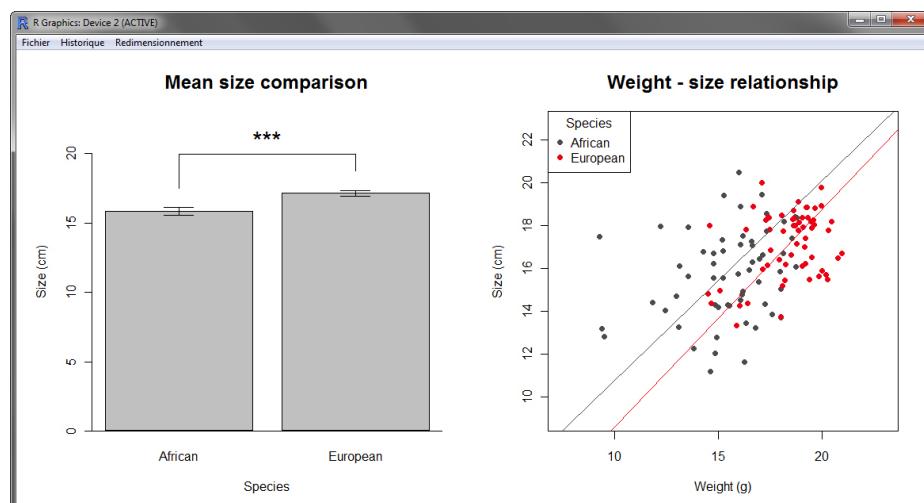


Figure 6: Final plots

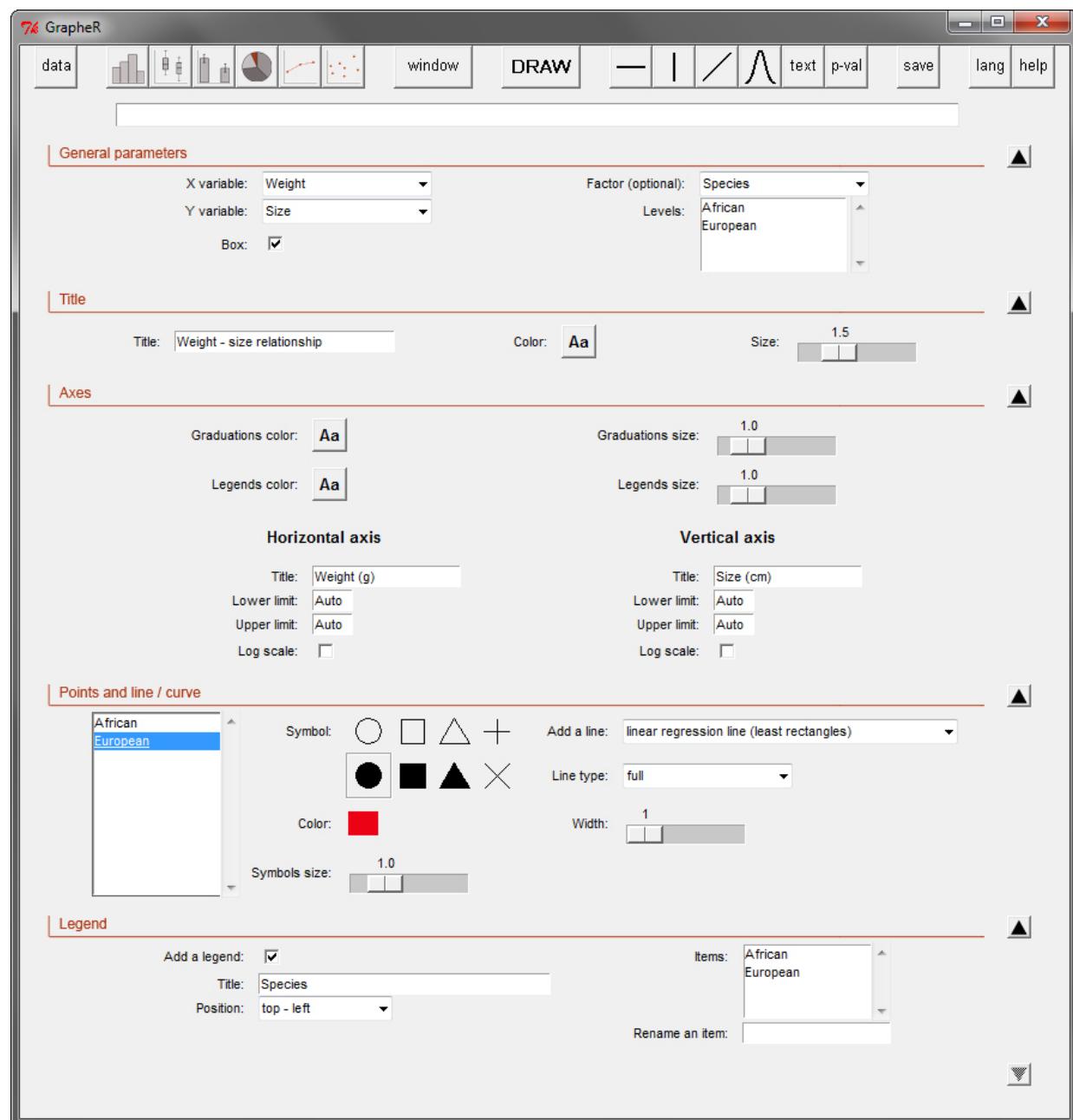


Figure 7: Illustrative scatter plot

Saving graph(s)

Once all graphs are drawn, they can be saved by clicking on the ‘Save’ button of the navigation bar. In the dialog box opening on the right of the interface, select the device containing the graph to be saved (Figure 8). Then choose the extension of the file to be created (‘jpg’, ‘png’ and ‘pdf’ are available) and the image length. Image height is automatically calculated from the length value. Finally click on the ‘Save’ button.

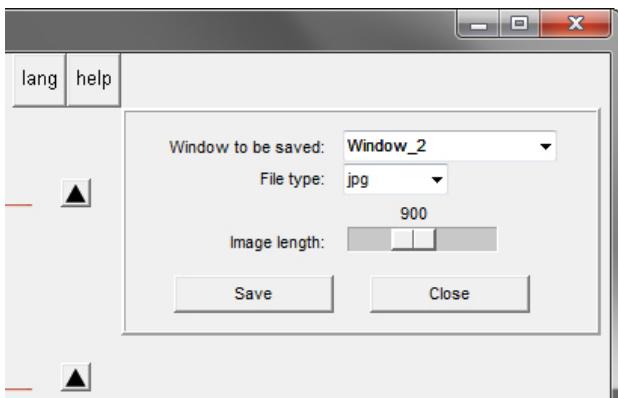


Figure 8: Saving graph(s)

Changing the user language

To change the user language, click on the ‘lang’ button of the navigation bar. A dialog box opens on the right of the interface (Figure 9). Choose the desired language in the drop-down menu. To fix your preference in the future, tick the ‘Save the preference’ checkbox. Click on the ‘Ok’ button to validate. The interface is closed and re-opened in the chosen language (but note that any dataset loaded in the previous language session is lost and has to be re-loaded).

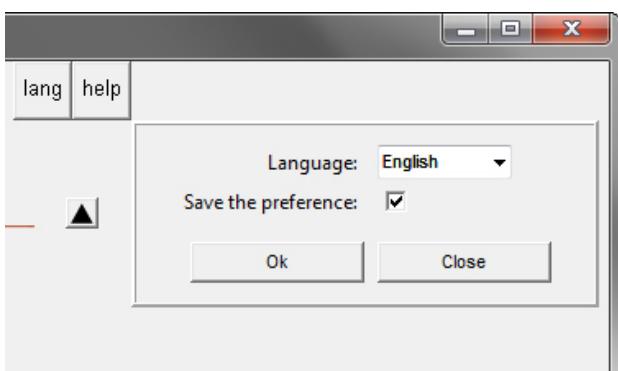


Figure 9: Changing the user language

Implementing GrapheR in another language

Implementing **GrapheR** in another language is very easy, because no word appearing in the interface is

written in the code. The button names, etc. come from an external file which is loaded depending on the language setting. In the current version (1.9-66), only English and French are available (files ‘Language_en.csv’ and ‘Language_fr.csv’ in the ‘lang’ directory).

Therefore, adding a new language just requires a (strict) translation of each line of one of the existing files (including spaces before and/or after words). The new file must then be saved as ‘Language_XX.csv’.

If you want to implement **GrapheR** in your language, you are most welcome. In that case, remember that it would be a good idea (but a tougher job) to also translate the user manual (contained in the ‘doc’ directory). If you want to participate, do not hesitate to contact me. I will take care of all screenshots to be included in the manual, and then add the link to the new language file in the code.

Acknowledgments

I am very grateful to Denis Poinsot, Dennis Webb, Clément Goubert and two anonymous referees for their precious advice on GUI ergonomics, **GrapheR** English translation and previous versions of this manuscript.

Bibliography

- F. Andrews. *playwith*: A GUI for interactive plots using GTK+. R package version 0.9-53. URL <http://code.google.com/p/playwith/>.
- P. Dalgaard. A Primer on the R-Tcl/Tk Package. *R News*, 1(3):27–31, 2001. URL <http://journal.r-project.org/>.
- P. Dalgaard. Changes to the R-Tcl/Tk Package. *R News*, 2(3):25–71, 2002. URL <http://journal.r-project.org/>.
- J. Fox. The R Commander: A Basic-Statistics Graphical User Interface to R. *Journal of Statistical Software*, 14(9), 2005. URL <http://www.jstatsoft.org/>.
- T. Gilliam and T. Jones. Monty Python Holy Grail. EMI Films, UK.
- P. Grosjean. *Sci-Views-R: A GUI API for R*. UMONS, Mons, Belgium, 2010. URL <http://www.sciviews.org/SciViews-R/>.
- M. Helbig, M. Theus, and S. Urbanek. JGR: Java GUI for R. *Statistical Computing and Graphics Newsletter*, 16(2):9–12, 2005. URL <http://stat-computing.org/newsletter/>.
- M. Hervé. GrapheR: A multiplatform GUI for drawing customizable graphs in R. R package version 1.9-66. URL <http://cran.r-project.org/package=GrapheR>.

R. Ihaka and R. Gentleman. R: A Language for Data Analysis and Graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996. URL <http://www.amstat.org/publications/jcgs/>.

G. J. Williams. Rattle: A Data Mining GUI for R. *The R Journal*, 1(2):45–55, 2009. URL <http://journal.r-project.org/>.

Maxime Hervé
UMR 1099 BiO3P (*Biology of Organisms and Populations applied to Plant Protection*)
University of Rennes 1 - Agrocampus Ouest - INRA
263 avenue du Général Leclerc, 35042 Rennes Cedex
France
mx.herve@gmail.com

rainbow: An R Package for Visualizing Functional Time Series

Han Lin Shang

Abstract Recent advances in computer technology have tremendously increased the use of functional data, whose graphical representation can be infinite-dimensional curves, images or shapes. This article describes four methods for visualizing functional time series using an R add-on package. These methods are demonstrated using age-specific Australian fertility data from 1921 to 2006 and monthly sea surface temperatures from January 1950 to December 2006.

Introduction

Recent advances in computer technology have enabled researchers to collect and store high-dimensional data. When the high-dimensional data are repeatedly measured over a period of time, a time series of functions can be observed. Although one can display high-dimensional time series by adapting multivariate techniques, it is important to take smoothness of functions into account (Ramsay and Dalzell, 1991). It is the smooth property of functions that separates functional time series from multivariate time series. Unlike longitudinal time series, functional time series mitigates the problem of missing data by an interpolation or smoothing technique, and so functional time series can be thought of as continuous.

Visualization aids the discovery of characteristics in data that might not have been apparent in mathematical models and summary statistics. Yet visualization has not yet received much attention in the literature of functional data analysis. Notable exceptions are the phase-plane plot of Ramsay and Ramsey (2002), which highlights important distributional characteristics using the first and second derivatives of functional data, and the singular value decomposition (SVD) plot of Zhang et al. (2007), which displays the changes in latent components in relation to the increases of the sample size or dimensionality. Another exception is the rainbow plot of Hyndman and Shang (2010), which can simultaneously display of functional data and identify possible outliers.

The aim of this article is to collect the R code that implements these graphical techniques. The R code of phase-plane plot is included in the **fda** package (Ramsay et al., 2011), while others are included in the **rainbow** package (Shang and Hyndman, 2011). In addition, this article also presents the use of animation, which can easily be used with all three graphical techniques in order to visualize time-varying features

of the data.

The article proceeds as follows. Visualization methods of functional time series are first reviewed. Illustrated by two data sets, the visualization methods are then demonstrated using the **rainbow** package. Conclusions are given in the end.

Data sets

The visualization methods are demonstrated using age-specific Australian fertility rates and monthly sea surface temperatures, both included in **rainbow**. The details of these two data sets are described below.

Figure 1 shows annual age-specific Australian fertility rates between ages 15 and 49 observed from 1921 to 2006. These data were obtained from the Australian Bureau of Statistics (Cat No. 3105.0.65.001, Table 38). The fertility rates are defined as the number of live births at 30th June each year, per 1000 of the female resident population of the same age.

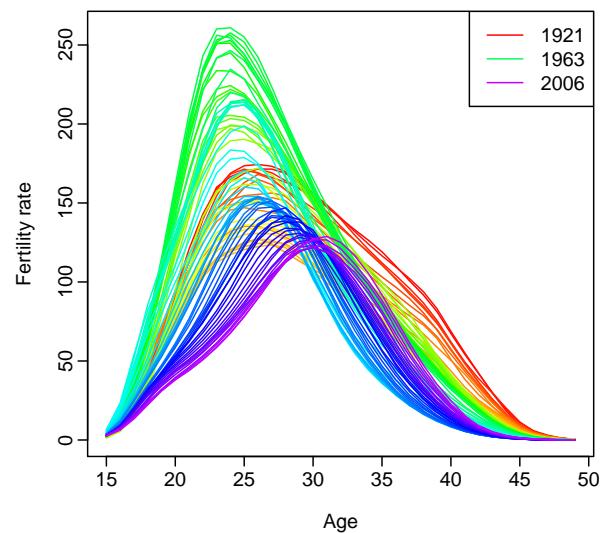


Figure 1: Smoothed Australian fertility rates between ages 15 and 49 observed from 1921 to 2006.

Although the four graphical techniques work equally well for plotting un-smoothed multivariate data, functional data ought to be smooth in nature. Therefore, the fertility rates were smoothed using a weighted median smoothing *B*-spline, constrained to be concave (see He and Ng, 1999; Hyndman and Ullah, 2007, for details).

Figure 2 shows monthly sea surface temperatures (in °C) from January 1950 to December 2006. These data were obtained from National Oceanic and Atmospheric Administration (<http://www.cpc.noaa.gov/data/indices/sstoi.indices>). These sea surface

temperatures were measured by moored buoys in the "Niño region", defined as the area within the coordinate $0 - 10^\circ$ South and $90 - 80^\circ$ West.

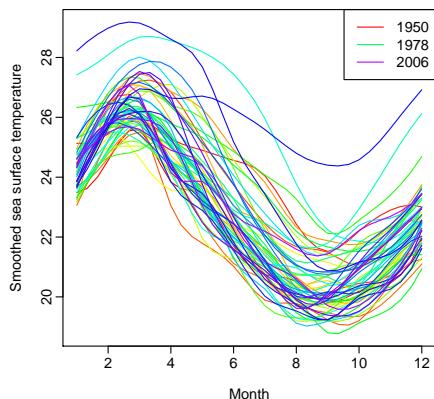


Figure 2: Smoothed monthly sea surface temperatures (in $^{\circ}\text{C}$) from January 1950 to December 2006.

The sea surface temperatures were smoothed using a smoothing spline with the smoothing parameter determined by generalized cross validation. Each curve represents smoothed sea surface temperatures in each year.

Functional time series visualization methods

Rainbow plot

The rainbow plot is a graphical display of all the functional data, with the only additional feature being a rainbow color palette based on an ordering of the data. By default, the rainbow plot displays functional data that are naturally ordered by time. Functional data can also be ordered by halfspace location depth (Tukey, 1975) and highest density regions (Hyndman, 1996). The depth and density orderings lead to the developments of functional bagplot and functional HDR boxplot, described in the next subsections.

As the referees pointed out, the rainbow plot (with the default rainbow color palette) may not be suitable for readers with color blindness. To mitigate this problem, the `plot.fds` function allows users to specify their preferred color, such as the heat or terrain palettes. In addition to the computer-screen based RGB colors, the `plot.fds` function allows users to utilize the perceptually-based Hue-Chroma-Luminance (HCL) colors included in the `colorspace` package (Ihaka et al., 2011). The use of HCL colors is superior to RGB colors for readability and color separation, and it is thus preferred (Zeileis et al., 2009).

Figure 1 presents the rainbow plot of the smoothed fertility rates in Australia between ages 15 and 49 observed from 1921 to 2006. The fertility rates from the distant past years are shown in red, while the most recent years are shown in violet. The peak of ferti-

lity rates occurred around 1961, followed by a rapid decrease during the 1980s, due to the increasing use of contraceptive pills. Then there is an increase in fertility rates at higher ages in the most recent years, which may be caused by a tendency to postpone child-bearing while pursuing careers. The rainbow plot is useful to reveal pattern changes for functional time series with a trend. It was produced by the following code:

```
# load the package used throughout this article
library("rainbow")
# plot.type = "function", curves are plotted by time
# the most recent curve is shown in purple
# the distant past curve is shown in red
plot(Australiasmoothfertility, plot.type = "functions",
      plotlegend = TRUE)
plot(ElNinosmooth, plot.type = "functions",
      plotlegend = TRUE)
```

For functional time series without a trend (e.g., Figure 2), the rainbow plot can still be used by constructing other order indexes, such as halfspace location depth and highest density regions. The colors of curves are then chosen in a rainbow color palette according to the ordering of depth or density.

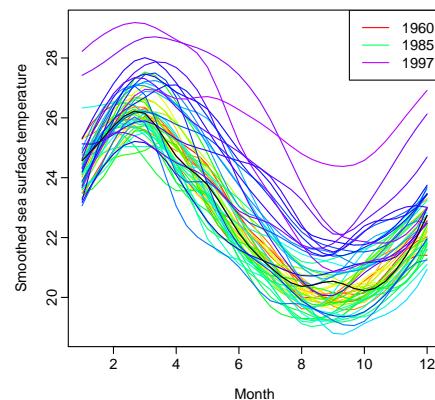


Figure 3: Rainbow plot with depth ordering. The median curve is shown in black.

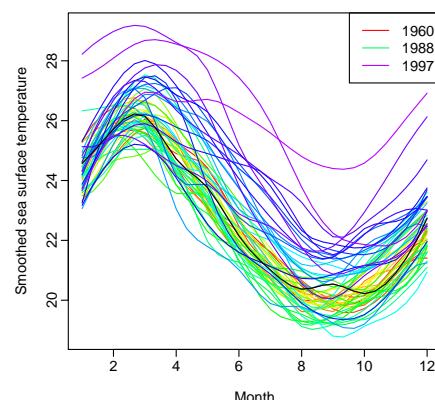


Figure 4: Rainbow plot with density ordering. The mode curve is shown in black.

Figures 3 and 4 present the rainbow plots of sea

surface temperatures ordered by halfspace location depth and highest density regions. The colors reflect the ordering and follow the order of the rainbow. The curves closest to the center of the data set are shown in red, whereas the most outlying curves are shown in violet. The curves are plotted in the order of depth and density, so the red curves are mostly obscured, but the violet curves are clearly seen even if they overlap with the majority of the data. These rainbow plots were produced using the following code.

```
# plot.type="depth", curves are plotted by depth
# depth is distance between median and each curve
# median curve (black line) is the center
plot(ElNinosmooth, plot.type = "depth",
     plotlegend = TRUE)

# plot.type="density", curves are plotted by density
# mode (black line) has the highest density
plot(ElNinosmooth, plot.type = "density",
     plotlegend = TRUE)
```

Functional bagplot

Adopting from the idea of projection pursuit (Cook et al., 1995), Hyndman and Shang (2010) use a robust functional principal component analysis to decompose functional data into the first two functional principal components and their principal component scores. As surrogates for functional data, the bivariate principal component scores can be ordered by Tukey's halfspace location depth and plotted in a familiar two-dimensional graph.

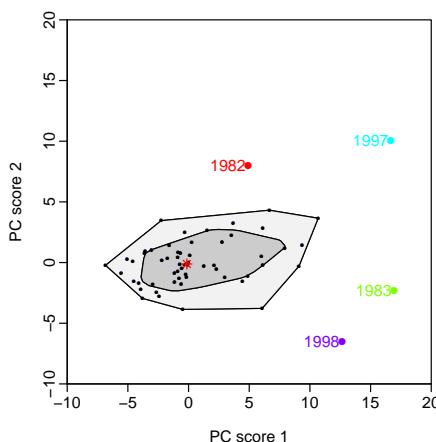


Figure 5: The bivariate bagplot.

Following Jones and Rice (1992) and Sood et al. (2009), the functional bagplot is considered as a mapping of the bivariate bagplot (Rousseeuw et al., 1999) of the first two robust principal component scores to the functional curves. The functional bagplot displays the median curve, and the inner and outer regions. The inner region is defined as the region bounded by all curves corresponding to the points in the bivariate bag. Hence, 50% of curves are in the inner region. The outer region is similarly defined as the

region bounded by all curves corresponding to the points within the bivariate fence region. The colors of bivariate outliers are matched to the same colors of functional outliers.

Figures 5 and 6 display the bivariate and functional bagplots of the sea surface temperature data.

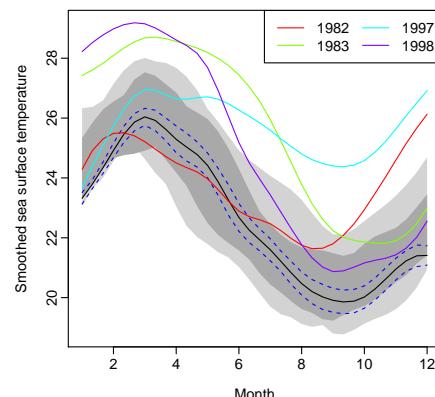


Figure 6: The functional bagplot.

The detected outliers in the sea surface temperature data are the years 1982-1983 and 1997-1998. The sea surface temperatures during 1982-1983 began in June 1982 with a moderate increase, then there were abnormal increases between September 1982 and June 1983 (Timmermann et al., 1999). The sea surface temperatures during 1997-1998 were also unusual: they became extremely warm in the latter half of 1997, and stayed high for the early part of 1998.

In Figure 5, the dark gray region shows the 50% bag, and the light gray region exhibits the customary 99% fence. These convex hulls correspond directly to the equivalent regions with similar colors and shading in the functional bagplot (in Figure 6). Points outside these regions are defined as outliers. The different colors for these outliers enable the functional outliers to be matched to the bivariate outliers. The red asterisk marks the Tukey median of the bivariate principal component scores, and the solid black curve shows the median curve. The dotted blue line in the functional bagplot gives 95% pointwise confidence intervals for the median curve. These bagplots were produced using the following code.

```
# plot.type = "bivariate", the bivariate principal
# component scores are displayed
# type = "bag" requests the bagplot
fboxplot(ElNinosmooth, plot.type = "bivariate",
          type = "bag", ylim = c(-10, 20), xlim = c(-10, 20))
# plot.type = "functional", the bivariate pc scores
# are matched to corresponding curves
fboxplot(ElNinosmooth, plot.type = "functional",
          type = "bag")
```

Functional highest density region (HDR) boxplot

The bivariate principal component scores can also be ordered by the highest density regions. The highest density regions are the quantiles of a two-dimensional Parzen-Rosenblatt kernel density estimate, where the bandwidths are chosen by a plug-in method (Hyndman, 1996). In comparison to a depth-measure approach, the density-measure approach is able to display multimodality if it is present in the data.

The functional HDR boxplot is a mapping of the bivariate HDR boxplot (Hyndman, 1996) of the first two robust principal component scores to the functional curves. The functional HDR boxplot displays the modal curve (i.e., the curve with the highest density), and the inner and outer regions. The inner region is defined as the region bounded by all the curves corresponding to the points inside the 50% bivariate HDR. Thus, 50% of curves are in the inner region. The outer region is similarly defined as the region bounded by all the curves corresponding to the points within the outer bivariate HDR. The colors of bivariate outliers are matched to the same colors of functional outliers.

Figures 7 and 8 display the bivariate and functional HDR boxplots of the sea surface temperature data set. As with any outlier detection methods, the coverage probability of the outer region needs to be pre-specified. If we set the coverage probability of the outer region to be 93%, then the outliers detected would match the results obtained by the bagplot. This indicates that these outliers are not only far from the median, but also have the lowest density.

In Figure 7, the dark and light gray regions show the 50% HDR and the 93% outer HDR, respectively. These correspond directly to the equivalent regions with similar colors and shading in the functional HDR boxplot (in Figure 8). Points outside these outer regions are identified as the outliers. The use of different colors for these outliers enables the functional outliers to match with the bivariate outliers. The red dot in the bivariate HDR boxplot marks the mode of bivariate principal component scores, and it corresponds to the solid black curve in the functional HDR boxplot.

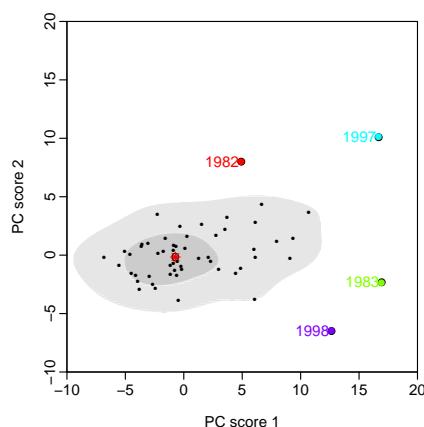


Figure 7: The bivariate HDR boxplot.

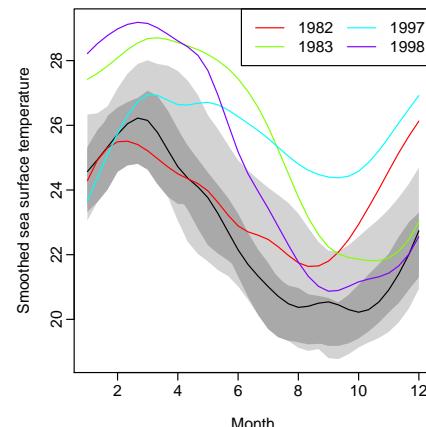


Figure 8: The functional HDR boxplot.

These HDR boxplots were produced using the following code.

```
# type = "hdr" requests the HDR boxplot
# alpha requests the coverage probability of inner
# and outer HDR regions, customarily c(0.05,0.5)
fboxplot(ElNinosmooth, plot.type = "bivariate",
         type = "hdr", alpha = c(0.07,0.5),
         ylim = c(-10,20), xlim = c(-10,20))
fboxplot(ElNinosmooth, plot.type = "functional",
         type = "hdr", alpha = c(0.07,0.5))
```

Singular value decomposition (SVD) plot

Zhang et al. (2007) proposed an interactive plot for visualizing patterns of functional data and multivariate data. They use the idea of projection pursuit to find only those low-dimensional projections that expose interesting features of the high-dimensional point cloud. As a popular projection pursuit technique, singular value decomposition (SVD) decomposes high-dimensional smoothed multivariate data into singular columns, singular rows, and singular values ordered by the amount of explained variance.

Zhang et al. (2007) discretize a set of functional data on a dense grid, denoted as $f(x_i) = [f_1(x_i), \dots, f_n(x_i)]'$, for $i = 1, \dots, p$, where p is the number of covariates, and n is the number of curves. Let $\{r_i; i = 1, \dots, p\}$ and $\{c_j; j = 1, \dots, n\}$ be the row and column vectors of the $(n \times p)$ matrix $f(x_i)$, respectively. The SVD of $f(x_i)$ is defined as

$$f(x_i) = s_1 u_1 v_1^T + s_2 u_2 v_2^T + \dots + s_K u_K v_K^T,$$

where the singular columns u_1, \dots, u_K form K orthonormal basis functions for the column space spanned by $\{c_j\}$; the singular rows v_1, \dots, v_K form K orthonormal basis functions for the row space spanned by $\{r_i\}$; and T symbolizes vector transpose. The vectors $\{u_k\}$ and $\{v_k\}$ are called singular column and singular row, respectively. The scalars s_1, \dots, s_K are called singular values. The matrix $\{s_k u_k v_k^T; k = 1, \dots, K\}$ is referred to as the SVD component.

Figure 9: The animation of SVD plot is started by clicking on any graph when using Adobe Reader.

The interactive plot of Zhang et al. (2007) captures the changes in the singular columns, as the number of curves gradually increases. Similarly, it also captures the changes in the singular rows, as the number of covariates gradually increases. The interactive plot simultaneously presents the column and row information of a two-way matrix, to relate the matrix to the corresponding curves, to show local variation, and to highlight interactions between columns and rows of a two-way matrix.

By using the **animate** package (Grahn, 2010), a set of dynamic movies can be created (see Figure 9 for an example). The advantage of creating these movies is the ability to demonstrate the time-varying features of the singular components, and to highlight outliers if present in the data. The animation is started by clicking on any graph when using Adobe reader.

Figure 9 shows the SVD plot of the sea surface temperature data set. The first SVD component captures the seasonal pattern, while the second and third SVD components show the contrasts of sea surface temperatures among different months. Note that the SVD2 and SVD3 are on a much smaller scale in comparison to the SVD1, because the SVD1 accounts for the most of the curves' variation. The functional time series can be approximated by the summation of the first three SVD components. From the residual plot, outliers can be identified if they differ significantly from zero.

The animated SVD plot was produced in a L^AT_EXeditor, such as WinEdt. For creating one ani-

mation in Figure 9, a series of 57 figures (as there are 57 years in the sea surface temperature data set) are linked together as follows

```
\% load a package in WinEdt
\usepackage{animate}
\% link a series of 57 figures for producing
\% one animation
\begin{figure}
  \animategraphics{figure_}{1}{57}
\end{figure}
```

In R, the non-animated SVD plot can be produced using the following code.

```
# order represents the number of SVD components
# as the number of SVD components increases
# the residuals should be centered around zero
# plot can be suppressed by setting plot = FALSE
SVDplot(ElNinosmooth, order = 3, plot = TRUE)
```

Conclusions

This article described four graphical methods included in the **rainbow** package, for visualizing functional time series. These methods can be categorized into graphical techniques using projection pursuit. Each of these methods has its unique advantages for revealing the characteristics of functional time series. Some of the methods enable us to identify and analyze abnormal observations, while others can be very useful in visualizing trend. Overall, these graphical

methods present a summary of functional time series, and should be considered as the first step of functional time series analysis.

Acknowledgements

Thanks to the editors and reviewers for constructive comments and suggestions that have significantly improved this article. Thanks also to Professor Rob Hyndman for helping with R code.

Bibliography

- D. Cook, A. Buja, J. Cabrera, and C. Hurley. Grand tour and projection pursuit. *Journal of Computational and Graphical Statistics*, 4(3):155–172, 1995. URL <http://www.jstor.org/stable/1390844>.
- A. Grahm. *The Animate Package*, 2010. URL <http://www.tug.org/texlive/Contents/live/texmf-dist/doc/latex/animate/animate.pdf>.
- X. He and P. Ng. COBS: qualitatively constrained smoothing via linear programming. *Computational Statistics*, 14(3):315–337, 1999. URL http://papers.ssrn.com/sol3/papers.cfm?abstract_id=185108.
- R. J. Hyndman. Computing and graphing highest density regions. *The American Statistician*, 50(2):120–126, 1996. URL <http://www.jstor.org/stable/2684423>.
- R. J. Hyndman and H. L. Shang. Rainbow plots, bagplots, and boxplots for functional data. *Journal of Computational and Graphical Statistics*, 19(1):29–45, 2010. URL <http://pubs.amstat.org/doi/pdf/10.1198/jcgs.2009.08158>.
- R. J. Hyndman and M. S. Ullah. Robust forecasting of mortality and fertility rates: A functional data approach. *Computational Statistics & Data Analysis*, 51(10):4942–4956, 2007. URL <http://portal.acm.org/citation.cfm?id=1241107.1241182>.
- R. Ihaka, P. Murrell, K. Hornik, and A. Zeileis. *colorspace: Color Space Manipulation.*, 2011. URL <http://CRAN.R-project.org/package=colorspace>. R package version 1.1-0.
- M. C. Jones and J. A. Rice. Displaying the important features of large collections of similar curves. *The American Statistician*, 46(2):140–145, 1992. URL <http://www.jstor.org/stable/2684184>.
- J. O. Ramsay and C. J. Dalzell. Some tools for functional data analysis (with discussion). *Journal of the Royal Statistical Society: Series B*, 53(3):539–572, 1991. URL <http://www.jstor.org/stable/2345586>.
- J. O. Ramsay and J. B. Ramsey. Functional data analysis of the dynamics of the monthly index of nondurable goods production. *Journal of Econometrics*, 107(1-2):327–344, 2002. URL <http://ideas.repec.org/a/eee/econom/v107y2002i1-2p327-344.html>.
- J. O. Ramsay, H. Wickham, S. Graves, and G. Hooker. *fda: Functional Data Analysis*, 2011. URL <http://CRAN.R-project.org/package=fda>. R package version 2.2.6.
- P. J. Rousseeuw, I. Ruts, and J. W. Tukey. The bagplot: a bivariate boxplot. *The American Statistician*, 53(4):382–387, 1999. URL <http://www.questia.com/googleScholar.qst?docId=5001888966>.
- H. L. Shang and R. J. Hyndman. *rainbow: Rainbow plots, bagplots and boxplots for functional data*, 2011. URL <http://CRAN.R-project.org/package=rainbow>. R package version 2.6.
- A. Sood, G. M. James, and G. J. Tellis. Functional regression: A new model for predicting market penetration of new products. *Marketing Science*, 28(1):36–51, 2009. URL <http://mktsci.journal.informs.org/cgi/content/abstract/mksc.1080.0382v1>.
- A. Timmermann, J. Oberhuber, A. Bacher, M. Esch, M. Latif, and E. Roeckner. Increased El Niño Frequency in a Climate Model Forced by Future Greenhouse Warming. *Nature*, 398(6729):694–697, 1999. URL <http://www.nature.com/nature/journal/v398/n6729/abs/398694a0.html>.
- J. W. Tukey. Mathematics and the picturing of data. In R. D. James, editor, *Proceedings of the International Congress of Mathematicians*, volume 2, pages 523–531. Canadian mathematical congress, Vancouver, 1975.
- A. Zeileis, K. Hornik, and P. Murrell. Escaping rgbland: selecting colors for statistical graphics. *Computational Statistics and Data Analysis*, 53(9):3259–3270, 2009. URL <http://www.sciencedirect.com/science/article/B6V8V-4VM43VH-1/2/b63f4a1a558083ab2b98f12e446d1ac7>.
- L. Zhang, J. S. Marron, H. Shen, and Z. Zhu. Singular value decomposition and its visualization. *Journal of Computational and Graphical Statistics*, 16(4):833–854, 2007. URL <http://pubs.amstat.org/doi/abs/10.1198/106186007X256080?journalCode=jcgs>

Han Lin Shang

Department of Econometrics and Business Statistics,
Monash University, Caulfield East, Victoria 3145,
Melbourne, Australia.

HanLin.Shang@monash.edu.au

Portable C++ for R Packages

by Martyn Plummer

Abstract Package checking errors are more common on Solaris than Linux. In many cases, these errors are due to non-portable C++ code. This article reviews some commonly recurring problems in C++ code found in R packages and suggests solutions.

CRAN packages are tested regularly on both Linux and Solaris. The results of these tests can be found at http://cran.r-project.org/web/checks/check_summary.html. Currently, 24 packages generate errors on Linux while 125 packages generate errors on Solaris.¹ A major contribution to the higher frequency of errors on Solaris is lack of portability of C++ code. The CRAN Solaris checks use the Oracle Solaris Studio 12.2 compiler, which has a much more stringent interpretation of the C++ standard than the GCC 4.6.1 compiler used for the checks on Linux, and will therefore reject code that compiles correctly with GCC.

It seems plausible that most R package developers work with GCC and are therefore not aware of portability issues in their C++ code until these are shown by the CRAN checks on Solaris. In fact, many of the testing errors are due to a few commonly recurring problems in C++. The aims of this article are to describe these problems, to help package authors diagnose them from the Solaris error message, and to suggest good practice for avoiding them.

The scope of the article is limited to basic use of C++. It does not cover the use of the **Rcpp** package (Eddelbuettel and Francois, 2011) and the Scythe statistical library (Pemstein et al., 2011), which are used to support C++ code in some R packages, nor issues involved in writing your own templates.

Before describing the portability issues in detail, it is important to consider two general principles that underlie most portability problems.

Firstly, C++ is not a superset of C. The current C standard is ISO/IEC 9899:1999, usually referred to as C99 after its year of publication. Most C++ compilers support the ISO/IEC 14882:1998 (C++98), which predates it.² Thus, the two languages have diverged, and there are features in C99 that are not available in C++98.

The g++ compiler allows C99 features in C++ code. These features will not be accepted by other compilers that adhere more closely to the C++98 standard. If your code uses C99 features, then it is not portable.

The C++ standard is evolving. In August 2011, the ISO approved a new C++ standard which was published in September 2011 and is known as C++11.

This should remove much of the divergence between the two languages. However, it may take some time for the new C++11 standard to be widely implemented in C++ compilers and libraries. Therefore this article was written with C++98 in mind.

The second general issue is that g++ has a permissive interpretation of the C++ standard, and will typically interpret ambiguous code for you. Other compilers require stricter conformance to the standard and will need hints for interpreting ambiguous code. Unlike the first issue, this is unlikely to change with the evolving C++ standard.

The following sections each describe a specific issue that leads to C++ portability problems. By far the most common error message produced on Solaris is ‘The function `foo` must have a prototype’. In the following, this is referred to as a *missing prototype* error. Problems and solutions are illustrated using C++ code snippets. In order to keep the examples short, ellipses are used in place of code that is not strictly necessary to illustrate the problem.

C99 functions

Table 1 shows some C functions that were introduced with the C99 standard and are not supported by C++98. These functions are accepted by g++ and will therefore pass R package checks using this compiler, but will fail on Solaris with a missing prototype error.

C99 Function	R replacement
<code>expm1(x)</code>	<code>expm1(x)</code>
<code>log1p(x)</code>	<code>log1p(x)</code>
<code>trunc(x)</code>	<code>ftrunc(x)</code>
<code>round(x)</code>	<code>fprec(x, 0)</code>
<code>lgamma(x)</code>	<code>lgammafn(x)</code>

Table 1: Some expressions using C99 functions and their portable replacements using functions declared in the ‘`<Rmath.h>`’

R packages have access to C functions exposed by the R API, which provides a simple workaround for these functions. All of the expressions in the left hand column of Table 1 can be replaced by portable expressions on the right hand side if the header ‘`<Rmath.h>`’ is included.

A less frequently used C99 function is the cube root function `cbrt`. The expression `cbrt(x)` can be replaced by `std::pow(x, (1./3.))` using the `pow` function defined in the header ‘`<cmath>`’.

¹Patched version of R 2.14.0, on 10 December 2011, x86 platform.

²Although a technical corrigendum of the C++ standard was published in 2003, it provided no new features.

C99 macros for special values

The C99 standard also introduced the constants `NAN` and `INFINITY` as well as the macros `isfinite`, `isinf`, `isnan` and `fpclassify` to test for them. None of these are part of the C++98 standard. Attempts to use the macros on Solaris will result in a missing prototype error, and the constants will result in the error message “`NAN/INFINITY not defined`”.

As with the C99 functions above, the R API provides some facilities to replace this missing functionality. The R macros `R_FINITE` and `ISNAN` and the R function `R_IsNaN` are described in the R manual “Writing R Extensions” and are accessed by including the header file ‘`<R.h>`’. They are not exactly equivalent to the C99 macros because they are adapted to deal with R’s missing value `NA_REAL`.

If you need access to a non-finite value then ‘`<R_ext/Arith.h>`’ provides `R_PosInf`, `R_NegInf` and `R_NaReal` (more commonly used as `NA_REAL`).

Variable-length arrays

A *variable-length array* is created when the size of the array is determined at runtime, not compile time. A simple example is

```
void fun(int n) {
    double A[n];
    ...
}
```

Variable length arrays are not part of the C++98 standard. On Solaris, they produce the error message “An integer constant expression is required within the array subscript operator”.

Variable-length arrays can be replaced by an instantiation of the `vector` template from the Standard Template Library (STL). Elements of an STL vector are accessed using square bracket notation just like arrays, so it suffices to replace the definition of the array with

```
std::vector<double> A(n);
```

The STL vector template includes a destructor that will free the memory when `A` goes out of scope.

A function that accepts a pointer to the beginning of an array can be modified to accept a reference to an STL vector. For example, this

```
void fun(double *A, unsigned int length) { ... }
```

may be replaced with

```
void fun(std::vector<double> &A) {
    unsigned int length = A.size();
    ...
}
```

Note that an STL vector can be queried to determine its size. Hence the size does not need to be passed as an additional argument.

External library functions that expect a pointer to a C array, such as BLAS or LAPACK routines, may also be used with STL vectors. The C++ standard guarantees that the elements of a vector are stored contiguously. The address of the first element (e.g. `&a[0]`) is thus a pointer to the start of an underlying C array that can be passed to external functions. For example:

```
int n = 20;
std::vector<double> a(n);
... // fill in a
double nrm2 = cblas_dnrm2(n, &a[0], 1);
```

Note however that boolean vectors are an exception to this rule. They may be packed to save memory, so it is not safe to assume a one-to-one correspondence between the underlying storage of a boolean vector and a boolean array.

Function overloading

The C++ standard library provides overloaded versions of most mathematical functions, with versions that accept (and return) a `float`, `double` or `long double`.

If an integer constant is passed to these functions, then g++ will decide for you which of the overloaded functions to use. For example, this expression is accepted by g++.

```
#include <cmath>
using std::sqrt;

double z = sqrt(2);
```

The Oracle Solaris Studio compiler will produce the error message ‘Overloading ambiguity between `std::sqrt(double)` and `std::sqrt(float)`’. It requires a hint about which version to use. This hint can be supplied by ensuring that the constant is interpreted as a double

```
double z = sqrt(2.);
```

In this case ‘`2.`’ is a double constant, rather than an integer, because it includes a decimal point. To use a `float` or `long double`, add the qualifying suffix `F` or `L` respectively.

The same error message arises when an integer variable is passed to an overloaded function inside an expression that does not evaluate to a floating point number.

```
#include <cmath>
using std::sqrt;

bool fun(int n, int m) {
    return n > m * sqrt(m);
}
```

In this example, the compiler does not know if `m` should be considered a `float`, `double` or `long double` inside the `sqrt` function because the return type is `bool`. The hint can be supplied using a cast:

```
return n > m * sqrt(static_cast<double>(m));
```

Namespaces for C functions

As noted above, C99 functions are not part of the C++98 standard. C functions from the previous C90 standard are allowed in C++98 code, but their use is complicated by the issue of namespaces. This is a common cause of missing prototype errors on Solaris.

The C++ standard library offers two distinct sets of header files for C90 functions. One set is called '`<cname>`' and the other is called '`<name.h>`' where "name" is the base name of the header (e.g. '`math`', '`stdio`', ...). It should be noted that the '`<name.h>`' headers in the C++ standard library are *not* the same files as their namesakes provided by the C standard library.

Both sets of header files in the C++ standard library provide the same declarations and definitions, but differ according to whether they provide them in the standard namespace or the global namespace. The namespace determines the function calling convention that must be used:

- Functions in the standard namespace need to be referred to by prefixing `std::` to the function name. Alternatively, in source files (but not header files) the directive using `std::foo`; may be used at the beginning of the file to instruct the compiler that `foo` always refers to a function in the standard namespace.
- Functions in the global namespace can be referred to without any prefix, except when the function is overloaded in another namespace. In this case the scope resolution prefix '`::`' must be used. For example:

```
using std::vector;

namespace mypkg {
    //Declare overloaded sqrt
    vector<double> sqrt(vector<double> const &);
    //Use sqrt in global namespace
    double y = ::sqrt(2.0);
}
```

Although the C++98 standard specifies which namespaces the headers '`<cname>`' and '`<name.h>`' should use, it has not been widely followed. In fact, the C++11 standard has been modified to conform to the current behaviour of C++ compilers ([JTC1/SC22/WG21 - The C++ Standards Committee, 2011, Appendix D.5](#)), namely:

- Header '`<cname>`' provides declarations and definitions in the namespace `std`. It may or may not provide them in the global namespace.
- Header '`<name.h>`' provides declarations and definitions in the global namespace. It may or may not provide them in the namespace `std`.

The permissiveness of this standard makes it difficult to test code for portability. If you use the '`<cname>`' headers, then `g++` puts functions in both the standard and global namespaces, so you may freely mix the two calling conventions. However, the Oracle Solaris Studio compiler will reject C function calls that are not resolved to the standard namespace.

The key to portability of C functions in C++ code is to use one set of C headers consistently and check the code on a platform that does not use both namespaces at the same time. This rules out `g++` for testing code with the '`<cname>`' headers. Conversely, the GNU C++ standard library header '`<name.h>`' does not put functions in the `std` namespace, so `g++` may be used to test code using '`<name.h>`' headers. This is far from an ideal solution. These headers were meant to simplify porting of C code to C++ and are not supposed to be used for new C++ code. However, the fact that the C++11 standard still includes these deprecated headers suggests a tacit acceptance that their use is still widespread.

Some `g++` shortcuts

The `g++` compiler provides two further shortcuts for the programmer which may occasionally throw up missing prototype errors on Solaris.

Headers in the GNU C++ standard library may be implicitly included in other headers. For example, in version 4.5.1, the '`<fstream>`' and '`<stdexcept>`' headers both include the '`<string>`' header. If your source file includes either of these two headers, then you may use strings without an `#include <string>;` statement, relying on this implicit inclusion. Other implementations of the C++ standard library may not do this implicit inclusion.

When faced with a missing prototype error on Solaris, it is worth checking a suitable reference (such as <http://www.cplusplus.com>) to find out which header declares the function according to the C++ standard, and then ensure that this header is explicitly included in your code.

A second shortcut provided by `g++` is *argument-dependent name lookup* (also known as *Koenig lookup*), which may allow you to omit scope resolution of functions in the standard namespace. For example,

```
#include <algorithm>
#include <vector>
using std::vector;
```

```
void fun (vector<double> &y)
{
    sort(y.begin(), y.end());
}
```

Since the arguments to the `sort` algorithm are in the `std` namespace, gcc looks in the same namespace for a definition of `sort`. This code therefore compiles correctly with gcc. Compilers that do not support Koenig lookup require the `sort` function to be resolved as `std::sort`.

Conclusions

The best way to check for portability of C++ code is simply to test it with as many compilers on as many platforms as possible. On Linux, various third-party commercial compilers are available at zero cost. The Intel C++ Composer XE compiler is free for non-commercial software development (Note that you must comply with Intel's definition of non-commercial); the PathScale EkoPath 4 compiler recently became open source; the Oracle Solaris Studio compilers may be downloaded for free from the Oracle web site (subject to license terms). The use of these alternative compilers should help to detect problems not detected by GCC, although it may not uncover all portability issues since they also rely on the GNU implementation of the C++ standard library. It is also possible to set up alternate testing platforms inside a virtual machine, although the details of this are beyond the scope of this article.

Recognizing that most R package authors do not have the time to set up their own testing platforms, this article should help them to interpret the feedback from the CRAN tests on Solaris, which provide a rigorous test of conformity to the current C++ standard. Much of the advice in this article also applies to a C++ front-end or an external library to which an R package may be linked.

An important limitation of this article is the assumption that package authors are free to modify the source code. In fact, many R packages are wrappers around code written by a third party. Of course, all CRAN packages published under an open source license may be modified according to the licence conditions. However, some of the solutions proposed here, such as using functions from the R API, may not be suitable for third-party code as they require maintaining a patched copy. Nevertheless, it may still be useful to send feedback to the upstream maintainer.

Acknowledgement

I would like to thank Douglas Bates and an anonymous referee for their helpful comments.

Bibliography

- D. Eddelbuettel and R. Francois. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 4 2011. ISSN 1548-7660. URL <http://www.jstatsoft.org/v40/i08>.
- JTC1/SC22/WG21 - The C++ Standards Committee. Working draft, standard for programming language C++. Technical report, ISO/IEC, February 2011. URL <http://www.open-std.org/JTC1/SC22/WG21>.
- D. Pemstein, K. M. Quinn, and A. D. Martin. The scythe statistical library: An open source C++ library for statistical computation. *Journal of Statistical Software*, 42(12):1–26, 6 2011. ISSN 1548-7660. URL <http://www.jstatsoft.org/v42/i12>.

Martyn Plummer

*International Agency for Research on Cancer
150 Cours Albert Thomas 69372 Lyon Cedex 08
France
plummerM@iarc.fr*

R's Participation in the Google Summer of Code 2011

by Claudia Beleites and John C. Nash

Abstract This article describes the activity of R in the 2011 Google Summer of Code (GSoC), which Wikipedia describes as follows.

The Google Summer of Code (GSoC) is an annual program, first held from May to August 2005, in which Google awards stipends (of 5000 USD, as of 2010) to hundreds of students who successfully complete a requested free or open source software coding project during the summer.

Essential elements of GSoC

Google has structured the GSoC to provide student resources for coding to open source software projects. While results can be used as part of a thesis or article, the focus is *coding*. Students receive a stipend of US\$ 5000 paid in 3 installments, which are only paid after each of the student's proposal, mid-term and final work has passed an evaluation. The open source projects, of which the R Foundation for Statistical Computing is one, are required to provide mentors for each student who makes a proposal, and the mentoring organization is given a gratuity of US\$ 500 per student who is funded.

The task of proposing coding projects falls to the mentoring organizations, and can be initiated by either prospective mentors or prospective students. Mentoring organizations "appoint" administrators for the GSoC activity. This year (2011) was the first we have had two people as co-admins. We can both attest that without this sharing, R's effort this year would very likely have foundered due to events in our lives that would have prevented critical tasks from being completed. The administrators shepherd the development of proposals from either mentors or students, try to encourage students to rework these proposals (the final proposals must come from the students), and guide the process of ranking them. Google has provided a web-based system (Melange) to manage the several thousand proposals, their ranking, and awarding of "slots" (funded students).

Google decides the number of slots each mentoring organization will receive. In 2008 and 2009, R got 4 slots, in 2010 we got 5, but in 2011 we were offered 15. The workload for the mentors and administrators grew similarly, as we had to rank approximately 40 proposals from which the funded 15 were chosen. It is possible for a student to propose more than one activity and be ranked highly by two

mentoring organizations. The administrators (including us) were therefore required to participate in an online chat "Deduplication" meeting, though there have not been such conflicts with the R GSoC student candidates.

Completed proposals in 2011

Sadly, one of our students did not carry out the proposed work but vanished for weeks without satisfactory explanation. This is not unknown; in the past 10–20% of GSoC students failed to complete their work (this year: 12%) (Google Summer of Code, 2011). We had one out of five in 2010. However, we did get 14 projects to a satisfactory state of completion this year. Seven of these related to graphical user interfaces (GUIs), images or visualization, four were related to optimization (two of these were also in the GUI/visualisation category), and one was related to OpenMP. We will very briefly summarize these, naming the students and mentors.

Andrew Rich: *Manipulating RStudio Graphics Towards Creating Intuitive Mathematical Comprehension*
(mentored by Daniel Kaplan & J. J. Allaire)
Extends the popular RStudio infrastructure to provide some interactive graphical tools useful in teaching mathematical and statistical concepts.

Sebastian Mellor: *Developing a hyperSpec GUI*
(mentored by Claudia Beleites and Colin Gillespie)
developed **hyperSpecGUI** with a graphical user interface to guide spike filtering for Raman spectra as demo application.

Xiaoyue Cheng: *Cranvastime: Interactive longitudinal and temporal data plots*
(mentored by Di Cook and Heike Hofmann)
is part of **cranvast**, interactive plots in R. Tools to explore for irregular periodicity by wrapping and folding time series, zoom and pan to focus details, link to other statistical plots and explore cross-correlation in multiple series, and individual patterns in longitudinal data.

Yixuan Qiu: *A GUI based package to assist optimization problems in R*
(mentored by John Nash and Ben Bolker)
The **optimgui** package from this project is directed towards the task of writing the objective function and related functions used by R **optim**

() and related tools. Users frequently find writing such objective functions is prone to errors or numerical instabilities.

Paula Moraga: *DClusterm: Model-based detection of disease clusters*

(mentored by Virgilio Gómez-Rubio and Barry Rowlingson)

produced package **DClusterm** implements several methods for the detection of disease clusters in spatial and spatio-temporal data.

The methods implemented in the package allow the use of GLM, GLMM and zero-inflated models for modelling the data.

Kåre Jacobsen: *Exploratory visualization of dynamic stochastic processes.*

(mentored by Niels Richard Hansen)

wrote **processDataAnim**.

Tuo Zhao: *HUGE: High-dimensional Undirected Graph Estimation*

(mentored by Kathryn Roeder and Han Liu)
produced package **huge**.

Sunil Kumar: *Image Analysis in R*

(mentored by Ian Fellows)
lead to **RImageJ**.

Lei Jiang: *OpenMP parallel framework for R*

(mentored by Ferdinand Jamitzky, George Ostrochov, and Pragneshkumar B. Patel)

Packages **ROpenMP** and **forpar** implement sequential-to-parallel source code transformation based on user-added OpenMP style directives (as comments in the code) using existing parallel packages in R and load the desired parallel backend in an on-the-fly manner for more ease in benchmarking, respectively.

Alexander Pillhöfer: *optile Category order optimization for graphical displays of categorical data*

(mentored by Antony Unwin)
produced package **extracat**.

Yu Chen: *TradeAnalytics toolchain enhancements*

(mentored by Brian Peterson)

The toolchain is used for forensic analysis of trades on financial markets.

Jennifer Feder Bobb: *Convergence acceleration of the Expectation-Maximization (EM) algorithms in computational statistics: A suite of cutting-edge acceleration schemes*

(mentored by Ravi Varadhan)

This work extended **SQUAREM** to build the **turboEM** package, for which different algorithms and documentation were developed in the following work.

Hui Zhao: *R-EM-Accelerator—Smarter Iterative Schemes Save Your Time*

(mentored by Roger Peng and Ravi Varadhan) implemented EM accelerator and benchmark examples in package **turboEM**.

Juemin Yang: *SpAM: an implementation of the Sparse Additive Models (SpAM)*

(mentored by Han Liu)

The package **SpAM** provides an implementation of the Sparse Additive Models (SpAM) and Greedy Sparse Additive Models (G-SpAM). Both the models are modern nonparametric predictive methods. They simultaneously conduct prediction (e.g. regression/classification), function estimation (e.g. smoothing and curve fitting), and feature selection. We targeted high-dimensional data analysis $d \gg n$.

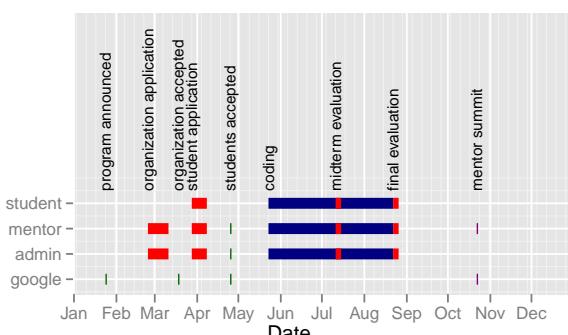
The packages can be accessed at <http://code.google.com/p/google-summer-of-code-2011-r/>.

Outcomes, benefits and costs

The main outcome of a GSoC student effort should be the code that becomes part of the mentoring organization's project, in our case, R packages. However, the short duration of the GSoC "Summer" means that it is more realistic to view the contribution as advancing the particular project of the student or mentor. An important adjunct to this is that the student is introduced into the developer community. In the longer term, we may see some impact from the code, as, for example, with the **roxygen** package. Furthermore, we have had GSoC students in one year return as mentors in another (e.g., I. Fellows, F. Schönbrodt).

The very presence of R in the GSoC list of mentoring organizations is a benefit in bringing student attention to the R Foundation and its work. The student stipends and modest gratuity are more concrete benefits.

There are considerable costs associated with these benefits of course. The mentors and administrators, as well as the students, are under a fairly tight timeline dictated by Google. This timeline is really only sensible in the context of the North American – possibly even Californian – academic calendar. Several of our 2011 students had major exams in the middle of their work period. The timeline is shown in the figure below.



Before the organization can apply to participate in GSoC, mentors have to prepare a statement of “project ideas”. Students can later respond to these or propose others, then after communicating with the mentors prepare a project proposal. Due to the tight and strict deadlines, mentors should start to develop ideas and possibly also to look out for students early (thin line in the time graph). It is strongly encouraged that the mentors pose some form of test or other evaluation of the capability of the student to do the coding.

Once the student proposals are submitted (this is done on the Melange system), the mentors must then rank them over a roughly two-week period. This is, in our opinion, one of the least pleasant aspects of GSoC. In some cases, we noted good projects from students that did not find a suitable mentor. And mentors really are the champions of particular proposals. We also saw several good proposals for one topic, and only weaker ones for another. Choosing was not easy.

Besides the quality of the proposal, more parameters for the selection were needed. While these could have been challenged by mentors, there was very little criticism of them. They were:

- To favour projects with a primary and alternate mentor over those with just one mentor.
- To allow only one project to be funded per primary mentor.
- To try to find a level of balance between topic areas.
- To accept complete proposals only.

As administrators, our most troubling issue was a mentor proposal that received almost unanimous criticism from other mentors as scientifically unsound and the project was subsequently rejected. Our main concern is less with an unhappy prospective mentor than with the student who put in effort to prepare a proposal that is rejected through misjudgement or misdirection of the mentor rather than bad quality of the student’s proposal. Possibly we could benefit from a separate committee to review mentor ideas before these are suggested for student consideration. However, the time line is very tight, and our resources are already very limited.

We managed communications for the collective of students and mentors with the Google Groups mailing list gsoc-r@googlegroups.com. For mentors-only communication, a separate mailing list gsoc-mentors@r-project.org is available. In addition, Melange automatically put students on a student GSoC mailing list, and mentors on the list google-summer-of-code-mentors-list@googlegroups.com. While traffic on the gsoc-r list was relatively light, the google-summer-of-code-mentors-list was quite “noisy”, with many trivial or non-GSoC items.

The Melange system Google chose to use for managing the GSoC effort is an in-house Google project, and is constantly being developed. This made it at times unstable. While Google staff was very friendly and replied promptly to our problems, mentors as well as students had to ask for assistance more than once.

Once the projects are going, we encouraged mentors and students to keep in regular communication. Some were lucky enough to be able to have face-to-face meetings. For the rest, we had email, Skype, and instant messaging, but for at least one of us, there were issues of time-zone differences which made it critical to agree when to communicate.

For the administrators, it was important to hear from the mentors and their students, and in this we are less satisfied. Essentially, we need a “status board” that has green, yellow and red lights to let us know which projects are going well, possibly in difficulty, or moribund respectively.

As mentioned, we had one “vanishing mentor” who turned up just in time for both mid-term and final evaluations, and this for a project with but one mentor. In that case, the mentor was having to relocate – ordinary life got in the way of GSoC. For the student, however, this is not a very good situation, and we were fortunate to have a student who was self-reliant and competent. In future, we would push harder to ensure that each project is well supported by at least two mentors. In addition we would wish the student had contacted us earlier when he did not get response from the mentor. The “status board” idea, however implemented, would be high on our priorities for such reasons.

We had one student who did very little work and did not reply for an extended period and was failed at the mid-term point. This is, of course, extremely unfortunate, but seems to be an anticipated occurrence in that the overall GSoC failure rate appears to be between 10 and 20% and related traffic at the GSoC mentors mailing list refers to it as a known issue. While it is recommended that proposals from mentors include a “test” to demonstrate student capability, there is little to assess reliability. Possibly proposals from students would have a higher chance of avoiding such drop-outs.

On the other hand, we were very happy when one of the high-ranked students told us promptly that he had got a proper job offer and would like to withdraw his proposal. This was early enough so that the slot could be awarded to another student.

Advice for future students, mentors and administrators

Our main piece of advice to all is to begin early. The project ideas, initial communications between students and mentors to organize good proposals, and

the preparation and refinement of student proposals all benefit from as much time as possible. In fact it is now the time to fill in the GSoC2012 project idea page at the R-Wiki (<http://rwiki.sciviews.org/doku.php?id=developers:projects:gsoc2012>).

Prospective mentors should arrange for backup in the form of a co-mentor. As administrators, we would recommend that the R GSoC effort not qualify any proposal without both a primary and backup mentor. Mentors should also recognize that the GSoC coding and a research practicum or thesis work are not the same. Our understanding is that Google regards the stipend as for coding, not as a general scholarship or bursary. The very name "Summer of Code" connotes a view that the coding is a form of summer employment. Mentors should also be aware that students can range in level from a first-year undergraduate to a PhD student about to defend.

As we have already suggested, we have particular concerns that students who make proposals based on mentors' ideas appear to have a large advantage over students with their own proposal. It is clear we need to find mentors for students with their own proposal. At the time when the proposals are being refined, it is late in the day to arrange suitable support and guidance for students who we cannot expect to be well-connected to our R community. Can we ask for volunteer potential mentors who are willing to

work with students who bring their own proposal? If so, we need to work out how to match such students with appropriate prospective mentors.

Next year's admins will be Toby Dylan Hocking and John C. Nash with Virgilio Gómez-Rubio as backup.

Bibliography

Google Summer of Code. ProgramStatistics: Program Information for Past Years. <http://code.google.com/p/google-summer-of-code/wiki/ProgramStatistics?ts=1315502962&updated=ProgramStatistics,11%202011>.

John C. Nash

*Telfer School of Management, University of Ottawa
55 Laurier E, Ottawa, ON, Canada, K1N 6N5
Canada*

nashjc@uottawa.ca

Claudia Beleites

*Department of Spectroscopy and Imaging
Institute of Photonic Technology
Albert-Einstein-Str. 9, 07745 Jena
Germany*

Claudia.Beleites@ipht-jena.de

Conference Report: useR! 2011

by Heather Turner

The seventh international R user conference, *useR! 2011*, took place at the University of Warwick, Coventry, 16–18 August 2011.

Following previous *useR!* conferences, this meeting of the R user community aimed to provide a platform for R users to discuss and exchange ideas of how R can be used for statistical computation, data analysis and visualization.

The conference attracted close to 450 participants, from 36 countries across North and South America, Europe, Africa, Asia and Australasia. The technical program comprised 136 regular talks, 30 lightning talks, two panel sessions, 21 regular posters and another 16 late-breaking posters, as well as eight invited talks. The social program consisted of an opening mixer, a poster reception sponsored by Revolution Analytics and a conference dinner sponsored by RStudio.

Pre-conference Tutorials

Prior to the conference, sixteen half-day tutorials were held. As in previous years, these tutorials were extremely popular, with 167 participants attending one or more of the following courses:

- *Douglas Bates*: Fitting and evaluating mixed models using **lme4**.
- *Roger Bivand and Edzer Pebesma*: Handling and analyzing spatio-temporal data in R.
- *Marine Cadoret and Sébastien Lê*: Analysing categorical data in R.
- *Stephen Eglen*: Emacs Speaks Statistics.
- *Andrea Foulkes*: High-dimensional data methods with R.
- *Frank E Harrell Jr*: Regression modeling strategies using the R package **rms**.
- *Søren Højsgaard*: Graphical models and Bayesian networks with R.
- *G. Jay Kerns*: Introductory probability and statistics using R.
- *Max Kuhn*: Predictive modeling with R and the **caret** package.
- *Fausto Molinari, Enrico Branca, Francesco De Filippo and Rocco Claudio Cannizzaro*: **R-Adamant**: Applied financial analysis & risk management.
- *Martin Morgan*: Bioconductor for the analysis of high-throughput genomic data.
- *Paul Murrell*: Introduction to grid graphics.
- *Giovanni Petris*: State space models in R.
- *Karline Soetaert and Thomas Petzoldt*: Simulating differential equation models in R.
- *Antony Unwin*: Graphical data analysis.
- *Brandon Whitcher, Jörg Polzehl, Karsten Tabelow*: Medical image analysis for MRI.

Invited Talks

The invited talks provided a focus point for the conference, where the participants gathered to hear from distinguished R users/developers. The logistical constraint of having to divide into two rooms, one with the speaker and one with a video link, seemed to pose little inconvenience to participants and overall there was a positive response to the following talks:

- *Adrian Bowman*: Modelling Three-dimensional Surfaces in R.
- *Lee Edlefsen*: High Performance Computing in R.
- *Ulrike Grömping*: Design of Experiments in R.
- *Wolfgang Huber*: From Genomes to Phenotypes.
- *Brian Ripley*: The R Development Process.
- *Jonathan Rougier*: Nomograms for visualising relationships between three variables.
- *Simon Urbanek*: R Graphics: Supercharged - Recent Advances in Visualization and Analysis of Large Data in R.
- *Brandon Whitcher*: Quantitative Analysis of Medical Imaging Data in R.

User-contributed Sessions

The themes of this year's Focus sessions were as follows:

- Computational Physics and Chemometrics
- Data Management
- Data Mining
- Development of R
- Dimensionality Reduction and Variable Selection
- Ecology and Ecological Modelling
- Finance
- Genomics and Bioinformatics
- High Performance Computing
- Hydrology and Soil Science
- Inference
- Interfaces
- Modelling Systems and Networks
- Multivariate Data
- Neuroscience
- Official and Social Statistics
- Population Genetics
- Process Optimization
- Programming
- Psychometrics
- Statistical Modelling
- Reporting Technologies and Workflows
- R in the Business World
- Spatio-Temporal Statistics
- Teaching
- Visualisation and Graphics

These themes were also represented in the poster session and in the seven kaleidoscope sessions that presented talks particularly suitable for a wider audience.

A new feature of the *useR! 2011* conference was the opportunity for participants to give a *Lightning Talk*, a 5-minute presentation on any R-related topic aimed particularly at those new to R. This resulted in three highly enjoyable sessions on the themes:

- Community and Communication
- Statistics and Programming
- Package Showcase

Participants seemed to appreciate this fast-paced introduction to a wide range of topics and it provided lots of scope for discussion as we moved on to dinner and the evening poster reception.

Organisers

Many thanks go to this year's program committee:

Ramón Díaz-Uriarte, John Fox, Romain François, Robert Gramacy, Paul Hewson, Torsten Hothorn, Kate Mullen, Brian Peterson, Thomas Petzoldt, Anthony Rossini, Barry Rowlingson, Carolin Strobl,

Stefan Theussl, Heather Turner, Hadley Wickham and Achim Zeileis.

and also the local organisers:

John Aston, Julia Brettschneider, David Firth, Ashley Ford, Ioannis Kosmidis, Tom Nichols, Elke Thönnes and Heather Turner.

Further Information

The *useR! 2011* website, <http://www.R-project.org/useR-2011/> provides a record of the conference. Where authors have made them available, slides are accessible via the online conference schedule. In addition videos of several of the invited talks have been put on the R-bloggers website: <http://www.r-bloggers.com/RUG/category/user-conference/>.

*Heather Turner
Statistics Department,
University of Warwick,
Coventry, UK
ht@heatherturner.net*

Forthcoming Events: useR! 2012

The eighth international R user conference *useR! 2012* will take place at Vanderbilt University, Nashville, Tennessee, USA, 12–15 June 2012. Following previous *useR!* conferences, this meeting of the R user community will

- focus on R as the ‘lingua franca’ of data analysis and statistical computing;
- provide a platform for R users to discuss and exchange ideas on how R can be used for statistical computation, data analysis, visualization and exciting applications in various fields;
- give an overview of the new features of the ever evolving R project.

The program comprises invited talks, user-contributed sessions, lightning talks, a poster session, and pre-conference tutorials and short courses.

Invited talks

The program of invited talks will represent the spectrum of interest from important technical developments to exciting applications of R, presented by experts in the field. The list of invited talks will be announced shortly.

User-contributed sessions

In the contributed sessions, presenters will share innovative and interesting uses of R, covering topics such as:

- Bayesian statistics
- Big data
- Bioinformatics
- Business analytics and financial modeling
- Chemometrics and computational physics
- Data mining
- Econometrics & finance
- Environmetrics & ecological modeling
- High performance computing
- Imaging
- Interfaces with other languages/software
- Machine learning
- Multivariate statistics
- Nonparametric statistics
- Pharmaceutical statistics
- Psychometrics
- Spatial statistics
- Statistics in the social and political sciences
- Teaching
- Visualization & graphics
- Web services with R

Abstracts may be submitted for oral or poster presentation. The poster session will be part of a major social

event on one of the evenings during the conference, or during the main program of talks. Submissions for contributed talks will be considered for the following types of session:

- *useR! Kaleidoscope*: These sessions give a broad overview of the many different applications of R and should appeal to a wide audience.
- *useR! Focus Session*: These sessions cover topics of special interest and may be more technical.

In addition to the main program of talks and based on their successful introduction at *useR! 2011*, all participants are invited to present a *Lightning Talk*, for which no abstract is required. These talks provide a 5-minute platform to speak on any R-related topic and should particularly appeal to those new to R. A variation of the *pecha kucha*¹ and *ignite*² formats will be used in which speakers must provide 15 slides to accompany their talk and each slide will be shown for 20 seconds.

The scientific program is organized by members of the program committee comprising Claudia Beleites, Dirk Eddelbuettel, John Fox, Benjamin French, Tal Galili, Jeffrey Horner, Andy Liaw, Uwe Ligges, Steve Miller, Kate Mullen, Bill Pikounis, Matt Shotwell, Heather Turner, Ravi Varadhan, and Achim Zeileis.

Pre-conference training

Tutorials Before the start of the official program, half-day tutorials will be offered on Tuesday, June 12th. The tutorials will address such topics as R for beginners, R for users of other statistical packages, R for statistical modeling, reproducible reporting with Sweave, web services with R, and graphics.

Short courses A new feature of *useR! 2012* is the full-day pre-conference short courses, to be held on Monday, June 11. These will include Frank Harrell’s *Regression Modeling Strategies and the rms Package* course.

Tutorial/short course presenters include Bill Venables, Terry Therneau, Robert Muenchen, Hadley Wickham, Doug Bates, Jeffrey Horner, Uwe Ligges, and Frank Harrell.

Location & surroundings

Participants will be well catered for at Vanderbilt University, with multiple nearby hotels as well as on-campus housing available. The University is near downtown Nashville which is close to a major airport.

¹http://en.wikipedia.org/wiki/Pecha_Kucha

²[http://en.wikipedia.org/wiki/Ignite_\(event\)](http://en.wikipedia.org/wiki/Ignite_(event))

Nashville is in middle Tennessee and is known as Music City USA for good reason, being the heart of country music, having a world-class symphony orchestra which has won 6 Grammy awards in the past 3 years, and hosting great jazz, bluegrass, blues, and many other types of music.

One of the largest music festivals in the US, *Bonnaroo*, takes place June 7–10 only 65 miles from Nashville. Another large festival, the country music *CMA Fest*, takes place in Nashville over the same days. The Cirque du Soleil show, *Michael Jackson THE IMMORTAL World Tour* comes to Nashville on June 12–13.

Further information

A web page offering more information on *useR! 2012*, including details regarding registration and abstract submission, is available at <http://www.R-project.org/useR-2012>.

We hope to meet you in Nashville!

The organizing committee:

Frank Harrell (chair), Stephania McNeal-Goddard, Matt Shotwell, JoAnn Alvarez, John Bock, Audrey Carvajal, WJ Cunningham, Chris Fonnesbeck, Tatsuki Koyama, Yanna Song, Sherry Stokes, Yuwei Zhu
useR-2012@R-project.org

Changes in R

From version 2.13.1 to version 2.14.0

by the R Core Team

CHANGES IN R VERSION 2.14.0

SIGNIFICANT USER-VISIBLE CHANGES

- All packages must have a namespace, and one is created on installation if not supplied in the sources. This means that any package without a namespace must be re-installed under this version of R (but previously-installed data-only packages without R code can still be used).
- The `yLineBias` of the `X11()` and `windows()` families of devices has been changed from 0.1 to 0.2: this changes slightly the vertical positioning of text in the margins (including axis annotations). This is mainly for consistency with other devices such as `quartz()` and `pdf()`. (Wish of PR#14538.)

There is a new graphics parameter "ylbias" which allows the y-line bias of the graphics device to be tweaked, including to reproduce output from earlier versions of R.

- Labeling of the p-values in various anova tables has been rationalized to be either "`Pr(>F)`" or "`Pr(>Chi)`" (i.e. the "`Pr(F)`", "`Pr(Chi)`" and "`P(>|Chi|)`" variants have been eliminated). Code which extracts the p value *via* indexing by name may need adjustment.
- `::` can now be used for datasets made available for lazy-loading in packages with namespaces (which makes it consistent with its use for data-only packages without namespaces in earlier versions of R).
- There is a new package **parallel**.

It incorporates (slightly revised) copies of packages **multicore** and **snow** (excluding MPI, PVM and NWS clusters). Code written to use the higher-level API functions in those packages should work unchanged (apart from changing any references to their namespaces to a reference to **parallel**, and links explicitly to **multicore** or **snow** on help pages).

It also contains support for multiple RNG streams following L'Ecuyer *et al* (2002), with support for both `mclapply` and **snow** clusters. This replaces functions like `clusterSetupRNG()` from **snow** (which are not in **parallel**).

The version released for R 2.14.0 contains base functionality: higher-level convenience func-

tions are planned (and some are already available in the 'R-devel' version of R).

- Building PDF manuals (for R itself or packages, e.g. *via* `R CMD check`) by default requires the LaTeX package 'inconsolata': see the section on 'Making the manuals' in the 'R Installation and Administration Manual'.
- `axTicks(*, log=TRUE)` has changed in some cases to satisfy the documented behavior and be consistent.

NEW FEATURES

- `txtProgressBar()` can write to an open connection instead of the console.
- Non-portable package names ending in '.' are no longer allowed. Nor are single-character package names (**R** was already disallowed).
- `regexpr()` and `gregexpr()` with `perl = TRUE` allows Python-style named captures. (Wish and contribution of PR#14518.)
- The placement of 'plotmath' text in the margins of plots done by base graphics now makes the same vertical adjustment as ordinary text, so using ordinary and plotmath text on the same margin line will seem better aligned (but not exactly aligned, since ordinary text has descenders below the baseline and plotmath places them on the baseline). (Related to PR#14537.)
- `sunflowerplot()` now has a formula interface. (Wish of PR#14541.)
- `iconv()` has a new argument `toRaw` to handle encodings such as UTF-16 with embedded nuls (as was possible before the CHARXP cache was introduced). It will also accept as input the type of list generated with `toRaw = TRUE`.
- Garbage-collecting an unused input text connection no longer gives a warning (since it 'connects' to nothing outside R).
- `read.table()` and `scan()` have gained a `text` argument, to allow reading data from a (possibly literal) character string.
- `optim(*, method = .)` now allows `method = "Brent"` as an interface to `optimize()`, for use in cases such as `mle()` where `optim()` is used internally.
- `mosaicplot()` gains a `border` argument. (Wish of PR#14550.)

- `smooth.spline()` gains a `tol` argument which controls how different `x` values need to be treated as distinct. The default has been changed to be more reliable for inputs whose range is small compared to their maximum absolute value. (Wish of PR#14452.)
- `g1()` runs faster by avoiding calling `factor()`.
- The `print()` method for `object.size()` accepts ‘B’ as well as ‘b’ as an abbreviation for ‘bytes’.
- `unlink()` gains a `force` argument to work like `rm -f` and if possible override restrictive permissions.
- `pbirthday()` and `qbirthday()` now use exact calculations for `coincident = 2`.
- `unzip()` and `unz()` connections have been updated with support for more recent Zip64 features (including large file sizes and `bzip2` compression, but not UTF-8 file names). `unzip()` has a new option to restore file times from those recorded (in an unknown timezone) in the zip file.
- `update.packages()` now accepts a character vector of package names for the `oldPkgs` argument. (Suggestion of Tal Galili.)
- The special reference class fields `.self` and `.refClassDef` are now read-only to prevent corrupting the object.
- `decompose()` now returns the original series as part of its value, so it can be used (rather than reconstructed) when plotting. (Suggestion of Rob Hyndman.)
- Rao’s efficient score test has been implemented for `glm` objects. Specifically, the `add1`, `drop1`, and `anova` methods now allow `test = "Rao"`.
- If a saved workspace (e.g. ‘.RData’) contains objects that cannot be loaded, R will now start with an warning message and an empty workspace, rather than failing to start.
- `strptime()` now accepts times such as ‘24:00’ for midnight at the end of the day, for although these are disallowed by POSIX 1003.1-2008, ISO 8601:2004 allows them.
- Assignment of `names()` to S4 objects now checks for a corresponding “names” slot, and generates a warning or an error if that slot is not defined. See the section on slots in `?Classes`.
- The default methods for `is.finite()`, `is.infinite()` and `is.nan()` now signal an error if their argument is not an atomic vector.
- The formula method for `plot()` no longer places package `stats` on the search path (it loads the namespace instead).
- There now is a genuine “function” method for `plot()` rather than the generic dispatching internally to `graphics::plot.function()`. It is now exported, so can be called directly as `plot.function()`.
- The one-sided `ks.test()` allows `exact = TRUE` to be specified in the presence of ties (but the approximate calculation remains the default: the ‘exact’ computation makes assumptions known to be invalid in the presence of ties).
- The behaviour of `curve(add = FALSE)` has changed: it now no longer takes the default `x` limits from the previous plot (if any): rather they default to `c(0, 1)` just as the “function” method for `plot()`. To get the previous behaviour use `curve(add = NA)`, which also takes the default for log-scaling of the `x-axis` from the previous plot.
- Both `curve()` and the `plot()` method for functions have a new argument `xname` to facilitate plots such as `sin(t) vs t`.
- The `local` argument to `source()` can specify an environment as well as `TRUE` (`parent.env()`) and `FALSE` (`.GlobalEnv`). It gives better error messages for other values, such as `NA`.
- `vcov()` gains methods for classes “`summary.lm`” and “`summary.glm`”.
- The `plot()` method for class “`profile.nls`” gains `ylab` and `lty` arguments, and passes ... on to `plot.default`.
- Character-string arguments such as the `mode` argument of `vector()`, `as.vector()` and `is.vector()` and the `description` argument of `file()` are required to be of length exactly one, rather than any further elements being silently discarded. This helps catch incorrect usage in programming.
- The `length` argument of `vector()` and its wrappers such as `numeric()` is required to be of length exactly one (other values are now an error rather than giving a warning as previously).
- `vector(len)` and `length(x) <- len` no longer accept `TRUE/FALSE` for `len` (not that they were ever documented to, but there was special-casing in the C code).
- There is a new function `Sys.setFileTime()` to set the time of a file (including a directory). See its help for exactly which times it sets on various OSes.

- The file times reported by `file.info()` are reported to sub-second resolution on systems which support it. (Currently the POSIX 2008 and FreeBSD/Darwin/NetBSD methods are detected.)
 - New function `getCall(m)` as an abstraction for `m$call`, enabling `update()`'s default method to apply more universally. (NB: this can be masked by existing functions in packages.)
 - `Sys.info()` gains a `euser` component to report the 'effective' user on OSes which have that concept.
 - The result returned by `try()` now contains the original error condition object as the "condition" attribute.
 - All packages with R code are lazy-loaded irrespective of the 'LazyLoad' field in the 'DESCRIPTION' file. A warning is given if the 'LazyLoad' field is overridden.
 - Rd markup has a new '\figure' tag so that figures can be included in help pages when converted to HTML or LaTeX. There are examples on the help pages for `par()` and `points()`.
 - The built-in `httpd` server now allows access to files in the session temporary directory `tempdir()`, addressed as the '/session' directory on the `httpd` server.
 - Development versions of R are no longer referred to by the number under which they might be released, e.g. in the startup banner, `R -version` and `sessionUtils()`. The correct way to refer to a development version of R is 'R-devel', preferably with the date and SVN version number.
E.g. 'R-devel (2011-07-04 r56266)'
 - There is a new function `texi2pdf()` in package **tools**, currently a convenience wrapper for `texi2dvi(pdf = TRUE)`.
 - There are two new options for typesetting PDF manuals from Rd files. These are 'beramono' and 'inconsolata', and used the named font for monospaced output. They are intended to be used in combination with 'times', and 'times,inconsolata,hyper' is now the default for the reference manual and package manuals. If you do not have that font installed, you can set `R_RD4PF` to one of the other options: see the 'R Installation and Administration Manual'.
 - Automatic printing for reference classes is now done by the `$show()` method. A method is defined for class 'envRefClass' and may be overridden for user classes (see the `?ReferenceClasses` example). S4 `show()` methods should no longer be needed for reference classes.
 - `tools::Rdiff` (by default) and R CMD `Rdiff` now ignore differences in pointer values when comparing printed environments, compiled byte code, etc.
 - The "source" attribute on functions created with `keep.source=TRUE` has been replaced with a "srcref" attribute. The "srcref" attribute references an in-memory copy of the source file using the "srcfilecopy" class or the new "srcfilealias" class.
 - New items "User Manuals" and **Technical Papers** have been added to the HTML help main page. These link to vignettes in the base and recommended packages and to a collection of papers about R issues, respectively.
 - Documentation and messages have been standardized to use "namespace" rather than "name space".
 - `setGeneric()` now looks in the default packages for a non-generic version of a function if called from a package with a namespace. (It always did for packages without a namespace.)
 - Setting the environment variable `_R_WARN_ON_LOCKED_BINDINGS_` will give a warning if an attempt is made to change a locked binding.
 - `\SweaveInput` is now supported when generating concordances in `Sweave()`.
 - `findLineNum()` and `setBreakpoint()` now allow the environment to be specified indirectly; the latter gains a `clear` argument to allow it to call `untrace()`.
 - The body of a closure can be one of further types of R objects, including environments and external pointers.
 - The `Rd2HTML()` function in package **tools** now has a `stylesheet` argument, allowing pages to be displayed in alternate formats.
 - New function `requireNamespace()` analogous to `require()`, returning a logical value after attempting to load a namespace.
 - There is a new type of RNG, "L'Ecuyer-CMRG", implementing L'Ecuyer (1999)'s 'combined multiple-recursive generator' 'MRG32k3a'. See the comments on `?RNG`.
 - `help.search()` and `??` can now display vignettes and demos as well as help pages. The new option "help.search.types" controls the types of documentation and the order of their display.
- This also applies to HTML searches, which now give results in all of help pages, vignettes and demos.

- `socketConnection()` now has a `timeout` argument. It is now documented that large values (package `snow` used a year) do not work on some OSes.
- The initialization of the random-number generator now uses the process ID as well as the current time, just in case two R processes are launched very rapidly on a machine with low-resolution wall clock (some have a resolution of a second; modern systems have microsecond-level resolution).
- New function `pskill()` in the **tools** package to send a terminate signal to one or more processes, plus constants such as `SIGTERM` to provide a portable way to refer to signals (since the numeric values are OS-dependent).
- New function `psnice()` in the **tools** package to return or change the ‘niceness’ of a process. (Refers to the ‘priority class’ on Windows.)
- `list.dirs()` gains a recursive argument.
- An ‘Authors@R’ field in a package ‘DESCRIPTION’ file can now be used to generate ‘Author’ and ‘Maintainer’ fields if needed, and to auto-generate package citations.
- New utility `getElement()` for accessing either a list component or a slot in an S4 object.
- `stars()` gains a `col.lines` argument, thanks to Dustin Sallings. (Wish of PR#14657.)
- New function `regmatches()` for extracting or replacing matched or non-matched substrings from match data obtained by `regexpr()`, `gregexpr()` and `regexec()`.
- `help(package = "pkg_name", help_type = "HTML")` now gives HTML help on the package rather than text help. (This gives direct access to the HTML version of the package manual shown via `help.start()`’s ‘Packages’ menu.)
- `agrep()` gains a `fixed` argument to optionally allow approximate regular expression matching, and a `costs` argument to specify possibly different integer match costs for insertions, deletions and substitutions.
- `read.dcf()` and `write.dcf()` gain a `keep.white` argument to indicate fields where whitespace should be kept as is.
- `available.packages()` now works around servers that fail to return an error code when ‘`PACKAGES.gz`’ does not exist. (Patch submitted by Seth Schommer.)
- `readBin()` can now read more than $2^{31} - 1$ bytes in a single call (the previously documented limitation).
- New function `regexec()` for finding the positions of matches as well as all substrings corresponding to parenthesized subexpressions of the given regular expression.
- New function `adist()` in package **utils** for computing ‘edit’ (generalized Levenshtein) distances between strings.
- Class “raster” gains an `is.na` method to avoid confusion from the misuse of the matrix method (such as PR#14618).
- The `identical()` function gains an `ignore.bytecode` argument to control comparison of compiled functions.
- `pmin` and `pmax` now warn if an argument is partially recycled (wish of PR#14638).
- The default for `image(useRaster=)` is now taken from option “preferRaster”: for the small print see `?image`.
- `str()` now displays reference class objects and their fields, rather than treating them as classical S4 classes.
- New function `aregexec()` in package **utils** for finding the positions of approximate string matches as well as all substrings corresponding to parenthesized subexpressions of the given regular expression.
- `download.file()` has an extra argument to pass additional command-line options to the non-default methods using command-line utilities.
`cacheOK = FALSE` is now supported for method = “curl”.
- `interaction.plot(*, type = .)` now also allows type “o” or “c”.
- `axTicks(*, log=TRUE)` did sometimes give more values than the ticks in the corresponding `graphics::axis()`. By default, it now makes use of the new (**graphics**-package independent) `axisTicks()` which can make use of a new utility `.axisPars()`. Further, it now returns a decreasing sequence (as for `log=FALSE`) when `usr` is decreasing.
- Using `fix()` or `edit()` on a R object (except perhaps a matrix or data frame) writes its temporary file with extension ‘.R’ so editors which select their mode based on the extension will select a suitable mode.

GRAPHICS DEVICES

- The `pdf()` device makes use of Flate compression: this is controlled by the new logical argument `compress`, and is enabled by default.

- Devices `svg()`, `cairo_pdf()` and `cairo_ps()` gain a `family` argument.

On a Unix-alike `X11()` gains a `family` argument. This is one of the `x11.options()` and so can be passed as an argument to the `bmp()`, `jpeg()`, `png()` and `tiff()` devices.

Analogous changes have been made on Windows, so all built-in R graphics devices now have a `family` argument except `pictex()` (which has no means to change fonts).

- The `bmp()`, `jpeg()`, `png()` and `tiff()` devices now make use of the `antialias` argument for `type = "quartz"`.

- There are several new built-in font mappings for `X11(type = "Xlib")`: see the help on `X11Fonts()`.

- There is a new type `X11(type = "dbcairo")` which updates the screen less frequently: see its help page.

- The `X11()` device now makes use of cursors to distinguish its states. The normal cursor is an arrow (rather than a crosshair); the crosshair is used when the locator is in use, and a watch cursor is shown when plotting computations are being done. (These are the standard names for X11 cursors: how they are actually displayed depends on the window manager.)

- New functions `dev.hold()` and `dev.flush()` for use with graphics devices with buffering. These are used for most of the high-level graphics functions such as `boxplot()`, so that the plot is only displayed when the page is complete.

Currently implemented for `windows(buffered = TRUE)`, `quartz()` and the `cairographics`-based `X11()` types with buffering (which are the default on-screen devices).

- New function `dev.capture()` for capture of bitmap snapshots of image-based devices (a superset of the functionality provided by `grid.cap()` in `grid`).

- The default `colormodel` for `pdf()` and `postscript()` is now called `"srgb"` to more accurately describe it. (Instead of `"rgb"`, and in the case of `postscript()` it no longer switches to and from the gray colorspace, by default.)

The `colormodel` for `postscript()` which does use both gray and sRGB colorspaces is now called `"srgb+gray"`.

Plots which are known to use only black/white/transparent can advantageously use `colormodel = "gray"` (just as before, but there is now slightly more advantage in doing so).

- `postscript()` with values `colormodel = "rgb"` and `colormodel = "rgb-nogray"` give the behaviour prior to R 2.13.0 of uncalibrated RGB, which under some circumstances can be rendered much faster by a viewer.

`pdf(colormodel = "rgb")` gives the behaviour prior to R 2.13.0 of uncalibrated RGB, which under some circumstances can be rendered faster by a viewer, and the files will be smaller (by about 9KB if compression is not used).

- The `postscript()` device only includes the definition of the sRGB colorspace in the output file for the colormodels which use it.

- The `postscript()` and `pdf()` devices now output greyscale raster images (and not RGB) when `colormodel = "gray"`.

- `postscript(colormodel = "gray")` now accepts non-grey colours and uses their luminance (as `pdf()` long has).

- `colormodel = "grey"` is allowed as an alternative name for `postscript()` and `pdf()`.

- `pdf()` in the default sRGB colorspace outputs many fewer changes of colorspace, which may speed up rendering in some viewing applications.

- There is a new function `dev.capabilities()` to query the capabilities of the current device. The initial set of capabilities are support for semi-transparent colours, rendering and capturing raster images, the locator and for interactive events.

- For `pdf()`, `maxRasters` is increased as needed so the argument is no longer used.

SWEAVE & VIGNETTES

- Options `keep.source = TRUE`, `figs.only = FALSE` are now the default.

- The way the type of user-defined options is determined has changed. Previously they were all regarded as logical: now the type is determined by the value given at first use.

- The allowed values of logical options are now precisely those allowed for character inputs to `as.logical()`: this means that 't' and 'f' are no longer allowed (although T and F still are).

- The preferred location for vignette sources is now the directory ‘vignettes’ and not ‘inst/doc’: R CMD build will now re-build vignettes in directory ‘vignettes’ and copy the ‘.Rnw’ (etc) files and the corresponding PDFs to ‘inst/doc’. Further files to be copied to ‘inst/doc’ can be specified *via* the file ‘vignettes/.install_extras’.
- R CMD Sweave now supports a ‘--driver’ option to select the Sweave driver: the default is equivalent to ‘--driver=RweaveLatex’.
- R CMD Sweave and R CMD Stangle support options ‘--encoding’ and ‘--options’.
- The Rtangle() driver allows output = “stdout” or output = “stderr” to select the output or message connection. This is convenient for scripting using something like

```
R CMD Stangle --options='output="stdout"' foo.Rnw > foo2.R
```

- There is a new option pdf.compress controlling whether PDF figures are generated using Flate compression (they are by default).
- R CMD Sweave now has a ‘--pdf’ option to produce a PDF version of the processed Sweave document.
- It is no longer allowed to have two vignettes with the same vignette basename (e.g. ‘vig.Rnw’ and ‘vig.Snw’). (Previously one vignette hid the other in the vignette() function.)

C-LEVEL FACILITIES

- Function R_tmpnam2 has been added to the API to allow a temporary filename to include a specified extension.

PACKAGE INSTALLATION

- Package ‘DESCRIPTION’ file field ‘KeepSource’ forces the package to be installed with keep.source = TRUE (or FALSE). (Suggestion of Greg Snow. Note that as all packages are lazy-loaded, this is now only relevant at installation.)

There are corresponding options ‘--with-keep.source’ and ‘--without-keep.source’ for R CMD INSTALL.

- R CMD INSTALL has a new option ‘--byte-compile’ to byte-compile the packages during installation (since all packages are now lazy-loaded). This can be controlled on a per-package basis by the optional field ‘ByteCompile’ in the ‘DESCRIPTION’ file.

- A package R code but without a ‘NAMESPACE’ file will have a default one created at R CMD build or R CMD INSTALL time, so all packages will be installed with namespaces. A consequence of this is that .First.lib() functions need to be copied to .onLoad() (usually) or .onAttach(). For the time being, if there is an auto-generated ‘NAMESPACE’ file and no .onLoad() nor .onAttach() function is found but .First.lib() is, it will be run as the attach hook (unless the package is one of a list of known exceptions, when it will be run as the load hook).
- A warning is given if test-loading a package changes a locked binding in a package other than itself. It is likely that this will be disallowed in future releases. (There are *pro tem* some exceptions to the warning.)
- A dependency on SVN revision is allowed for R, e.g. R (>= r56550). This should be used in conjunction with a version number, e.g. R (>= 2.14.0), R (>= r56550) to distinguish between R-patched and R-devel versions with the same SVN revision.
- installed.packages() now hashes the names of its cache files to avoid very rare problems with excessively long path names. (PR#14669)
- A top-level ‘COPYING’ file in a package is no longer installed (file names ‘LICENSE’ or ‘LICENCE’ having long been preferred).

UTILITIES

- R CMD check now gives an error if the R code in a vignette fails to run, unless this is caused by a missing package.
- R CMD check now unpacks tarballs in the same way as R CMD INSTALL, including making use of the environment variable R_INSTALL_TAR to override the default behaviour.
- R CMD check performs additional code analysis of package startup functions, and notifies about incorrect argument lists and (incorrect) calls to functions which modify the search path or inappropriately generate messages.
- R CMD check now also checks compiled code for symbols corresponding to functions which might terminate R or write to ‘stdout’/‘stderr’ instead of the console.
- R CMD check now uses a pdf() device when checking examples (rather than postscript()).
- R CMD check now checks line-endings of makefiles and C/C++/Fortran sources in subdirectories of ‘src’ as well as in ‘src’ itself.

- R CMD check now reports as a NOTE what look like methods documented with their full names even if there is a namespace and they are exported. In almost all cases they are intended to be used only as methods and should use the ‘\method’ markup. In the other rare cases the recommended form is to use a function such as `coefHclust` which would not get confused with a method, document that and register it in the ‘NAMESPACE’ file by `s3method(coef, hclust, coefHclust)`.
- The default for the environment variable `_R_CHECK_COMPACT_DATA2_` is now true: thus if using the newer forms of compression introduced in R 2.10.0 would be beneficial is now checked (by default).
- Reference output for a vignette can be supplied when checking a package by R CMD check: see ‘Writing R Extensions’.
- R CMD Rd2dvi allows the use of LaTeX package ‘inputenx’ rather than ‘inputenc’: the value of the environment variable `RD2DVI_INPUTENC` is used. (LaTeX package ‘inputenx’ is an optional install which provides greater coverage of the UTF-8 encoding.)
- Rscript on a Unix-alike now accepts file names containing spaces (provided these are escaped or quoted in the shell).
- R CMD build on a Unix-alike (only) now tries to preserve dates on files it copies from its input directory. (This was the undocumented behaviour prior to R 2.13.0.)

DEPRECATED AND DEFUNCT

- `require()` no longer has a `save` argument.
- The `gamma` argument to `hsv()`, `rainbow()`, and `rgb2hsv()` has been removed.
- The ‘`--no-docs`’ option for R CMD build -binary is defunct: use ‘`--install-args`’ instead.
- The option ‘`--unsafe`’ to R CMD INSTALL is defunct: use the identical option ‘`--no-lock`’ instead.
- The entry point `pythag` formerly in ‘`Rmath.h`’ is defunct: use instead the C99 function `hypot`.
- R CMD build -binary is formally defunct: R CMD INSTALL -build has long been the preferred alternative.
- `zip.file.extract()` is now defunct: use `unzip()` or `unz()` instead.

- R CMD Rd2dvi without the ‘`--pdf`’ option is now deprecated: only PDF output will be supported in future releases (since this allows the use of fonts only supported for PDF), and only R CMD Rd2pdf will be available.
- Options such as `-max-nsize` and the function `mem.limits()` are now deprecated: these limits are nowadays almost never used, and are reported by `gc()` when they are in use.
- Forms like `binomial(link = "link")` for GLM families deprecated since R 2.4.0 are now defunct.
- The `declarativeOnly` argument to `loadNamespace()` (not relevant since R 2.13.0) has been removed.
- Use of `library.dynam()` without specifying all the first three arguments is deprecated. (It is often called from a namespace, and the defaults are only appropriate to a package.)

Use of `chname` in `library.dynam()` with the extension ‘.so’ or .dll (which is clearly not allowed according to the help page) is deprecated. This also applies to `library.dynam.unload()` and `useDynLib` directives in NAMESPACE files.

- It is deprecated to use `mean(x)` and `sd(x)` directly on data frames (or also matrices, for `sd`) `x`, instead of simply using `apply`.
In the same spirit, `median(x)` now gives an error for a data frame `x` (it often gave nonsensical results).
- The `keep.source` argument to `library()` and `require()` is deprecated: it was only used for packages installed without lazy-loading, and now all packages are lazy-loaded.
- Using a false value for the ‘DESCRIPTION’ field ‘LazyLoad’ is deprecated.

INSTALLATION

- The base and recommended packages are now byte-compiled (equivalent to `make bytecode` in R 2.13.x).
- Configure option ‘`--with-system-zlib`’ now only makes use of the basic interface of zlib and not the C function ‘`gzseek`’ which has shown erroneous behaviour in zlib 1.2.4 and 1.2.5.
- The zlib in the R sources is now version 1.2.5. (This is safe even on 32-bit Linux systems because only the basic interface is now used.)
- The ‘.afm’ files in package `grDevices` are now installed as compressed files (as long done on Windows), saving ca 2MB on the installed size.

- The non-screen cairo-based devices are no longer in the X11 module and so can be installed without X11. (We have never seen a Unix-alike system with cairographics installed but not X11, but a user might select '--without-x'.)
- Configure will try to use -fobjc-exceptions for the Objective-C compiler (if present) to ensure that even compilers that do not enable exceptions by default (such as vanilla gcc) can be used. (Objective-C is currently only used on Mac OS X.)
- The system call `times` is required.
- The C99 functions `acosh`, `asinh`, `atanh`, `snprintf` and `vsnprintf` are now required.
- There is no longer support for making DVI manuals via `make dvi`, `make install-dvi` and similar. Only PDF manuals are supported (to allow the use of fonts which are only available for PDF.)
- The 'configure' arguments used during configuration of R are included as a comment in 'Makeconf' for informative purposes on Unix-alikes in a form suitable for shell execution. Note that those are merely command-line arguments, they do not include environment variables (one more reason to use configure variables instead) or site configuration settings.
- Framework installation now supports `DESTDIR` (Mac OS X only).
- Java detection (`R CMD javareconf`) works around bogus `java.library.path` property in recent Oracle Java binaries.

BUG FIXES

- The locale category 'LC_MONETARY' was only being set on startup on Windows: that is now done on Unix-alikes where supported.
- Reference class utilities will detect an attempt to modify methods or fields in a locked class definition (e.g., in a namespace) and generate an error.
- The formula methods for `lines()`, `points()` and `text()` now work even if package `stats` is not on the search path.
- In principle, S4 classes from different packages could have the same name. This has not previously worked. Changes have now been installed that should allow such classes and permit methods to use them. New functions `className()` and `multipleClasses()` are related tools for programming.

- Work around an issue in Linux (a system `select` call resetting `tv`) which prevented internet operations from timing out properly.
- Several stack trampling and overflow issues have been fixed in TRE, triggered by `agrep` and friends with long patterns. (PR#14627.)
- ("design infelicity") Field assignments in reference classes are now consistent with slots in S4 classes: the assigned value must come from the declared class (if any) for the field or from a subclass.
- The methods objects constructed for "coerce" and "coerce<-" were lacking some essential information in the generic, defined and target slots; `as()` did not handle duplicate class definitions correctly.
- The parser no longer accepts the digit 8 in an octal character code in a string, nor does it accept unterminated strings in a file. (Reported by Bill Dunlap.)
- The `print()` method for class "summary.aov" did not pass on argument `digits` when `summary()` was called on a single object, and hence used more digits than documented.
- The `X11()` device's cairo back-end produced incorrect capture snapshot images on big-endian machines.
- `loglin()` gave a spurious error when argument `margin` consisted of a single element of length one. (PR#14690)
- `loess()` is better protected against misuse, e.g. zero-length span. (PR#14691)
- `HoltWinters()` checks that the optimization succeeded. (PR#14694)
- The (undocumented) inclusion of superclass objects in default initializing of reference classes overwrote explicit field arguments. The bug is fixed, the feature documented and a test added.
- `round(x, -Inf)` now does something sensible (return zero rather than `NA`).
- `signif(x, -Inf)` now behaves as documented (`signif(x, 1)`) rather than giving zero.
- The "table" method for `Axis()` hardcoded `side = 1`, hence calls to `plot(<vector>, <table>)` labelled the wrong axis. (PR#14699)
- Creating a connection might fail under `gctorture(TRUE)`.

- `stack()` and `unstack()` converted character columns to factors.
`unstack()` sometimes produced incorrect results (a list or a vector) if the factor on which to un-split had only one level.
- On some systems `help(".C", help_type = "pdf")` and similar generated file names that TeX was unable to handle.
- Non-blocking listening socket connections continued to report `isIncomplete()` as true even when the peer had closed down and all available input had been read.
- The revised HTML search system now generates better hyperlinks to help topics found: previously it gave problems with help pages with names containing e.g. spaces and slashes.
- A late change in R 2.13.2 broke '`\Sexpr`' expressions in Rd files.
- The creation of ticks on log axes (including `axTicks()`) sometimes incorrectly omitted a tick at one end
- The creation of ticks on log axes (including by `axTicks()`) sometimes incorrectly omitted a tick at one end of the range by rounding error in a platform-dependent way. This could be seen in the examples for `axTicks()`, where with axis limits `c(0.2, 88)` the tick for 0.2 was sometimes omitted.
- `qgamma()` for small shape underflows to 0 rather than sometimes giving `NaN`. (PR#8528, PR#14710)
- `mapply()` now gives an explicit error message (rather than an obscure one) if inputs of zero and positive length are mixed.
- Setting a Hershey font family followed by string height query would crash R.
- R CMD `javareconf -e` would fail for some shells due to a shift error. Also the resulting paths will no longer contain `$JAVA_HOME` as that can result in an unintended substitution based on 'Makeconf' instead of the shell setting.

CHANGES IN R VERSION 2.13.2

NEW FEATURES

- `mem.limits()` now reports values larger than the maximum integer (previously documented to be reported as `NA`), and allows larger values to be set, including `Inf` to remove the limit.

- The `print()` methods for classes "Date", "POSIXct" and "POSIXlt" respect the option "`max.print`" and so are much faster for very long datetime vectors. (Suggestion of Yohan Chalabi.)
- `untar2()` now works around errors generated with tar files that use more than the standard 6 digits for the checksum. (PR#14654)
- `install.packages()` with `Ncpus > 1` guards against simultaneous installation of indirect dependencies as well as direct ones.
- Sweave now knows about a few more Windows' encodings (including `cp1250` and `cp1257`) and some `inputenx` encodings such as `koi8-r`.
- `postscript(colormodel = "rgb-nogray")` no longer sets the sRGB colorspace for each colour and so some viewers may render its files much faster than the default `colormodel = "rgb"`.
- The default for `pdf(maxRasters=)` has been increased from 64 to 1000.
- `readBin()` now warns if `signed = FALSE` is used inappropriately (rather than being silently ignored).
It enforces the documented limit of $2^{31} - 1$ bytes in a single call.
- PCRE has been updated to version 8.13, a bug-fix release with updated Unicode tables (version 6.0.0). An additional patch (r611 from PCRE 8.20-to-be) has been added to fix a collation symbol recognition issue.

INSTALLATION

- It is possible to build in 'src/extra/xdr' on more platforms. (Needed since glibc 2.14 hides its RPC implementation.)
- `configure` will find the Sun TI-RPC implementation of `xdr` (in 'libtirpc') provided its header files are in the search path: see the 'R Installation and Administration Manual'.

PACKAGE INSTALLATION

- Using a broad `exportPattern` directive in a 'NAMESPACE' file is no longer allowed to export internal objects such as `.onLoad` and `__S3MethodsTable__`.
These are also excluded from imports, along with `.First.lib`.

BUG FIXES

- `fisher.test()` had a buglet: If arguments were factors with unused levels, levels were dropped and you would get an error saying that there should be at least two levels, inconsistently with pre-tabulated data. (Reported by Michael Fay).
- `package.skeleton()` will no longer dump S4 objects supplied directly rather than in a code file. These cannot be restored correctly from the dumped version.
- Build-time expressions in help files did not have access to functions in the package being built (with R CMD build).
- Because `quote()` did not mark its result as being in use, modification of the result could in some circumstances modify the original call.
- Plotting `pch = '.'` now guarantees at least a one-pixel dot if `cex > 0`.
- The very-rarely-used command-line option '`--max-vsize`' was incorrectly interpreted as a number of Vcells and not in bytes as documented. (Spotted by Christophe Rhodes.)
- The HTML generated by `Rd2HTML()` comes closer to being standards compliant.
- `filter(x, recursive = TRUE)` gave incorrect results on a series containing NAs. (Spotted by Bill Dunlap.)
- Profiling `stats:::mle()` fits with a fixed parameter was not supported. (PR#14646)
- `retracemem()` was still using positional matching. (PR#14650)
- The `quantile` method for "ecdf" objects now works and is documented.
- `xtabs(~ ... , sparse=TRUE)` now also works together with an `exclude = ..` specification.
- `decompose()` computed an incorrect seasonal component for time series with odd frequencies.
- The `pdf()` device only includes the definition of the sRGB colorspace in the output file for the "rgb" colormodel (and not for "gray" nor "cmyk"): this saves ca 9KB in the output file.
- `.hasSlot()` wrongly gave FALSE in some cases.
- `Sweave()` with `keep.source=TRUE` could generate spurious NA lines when a chunk reference appeared last in a code chunk.

- '`\Sexpr[results=rd]`' in an '`.Rd`' file now first tries `parse_Rd(fragment=FALSE)` to allow Rd section-level macros to be inserted.
- The `print()` method for class "`summary.aov`" did not pass on arguments such as `signif.stars` when `summary()` was called on a single object. (PR#14684)
- In rare cases `ks.test()` could return a p-value very slightly less than 0 by rounding error. (PR#14671)
- If `trunc()` was called on a "`POSIXlt`" vector and the result was subsetted, all but the first element was converted to NA. (PR#14679)
- `cbind()` and `rbind()` could cause memory corruption when used on a combination of raw and logical/integer vectors.

CHANGES IN R VERSION 2.13.1

NEW FEATURES

- `iconv()` no longer translates NA strings as "NA".
- `persp(box = TRUE)` now warns if the surface extends outside the box (since occlusion for the box and axes is computed assuming the box is a bounding box). (PR#202.)
- `RShowDoc()` can now display the licences shipped with R, e.g. `RShowDoc("GPL-3")`.
- New wrapper function `showNonASCIIfile()` in package `tools`.
- `nobs()` now has a "`mle`" method in package `stats4`.
- `trace()` now deals correctly with S4 reference classes and corresponding reference methods (e.g., `$trace()`) have been added.
- `xz` has been updated to 5.0.3 (very minor bugfix release).
- `tools:::compactPDF()` gets more compression (usually a little, sometimes a lot) by using the compressed object streams of PDF 1.5.
- `cairo_ps(onefile = TRUE)` generates encapsulated EPS on platforms with `cairo >= 1.6`.
- Binary reads (e.g. by `readChar()` and `readBin()`) are now supported on clipboard connections. (Wish of PR#14593.)
- `as.POSIXlt.factor()` now passes ... to the character method (suggestion of Joshua Ulrich). [Intended for R 2.13.0 but accidentally removed before release.]

- `vector()` and its wrappers such as `integer()` and `double()` now warn if called with a length argument of more than one element. This helps track down user errors such as calling `double(x)` instead of `as.double(x)`.

INSTALLATION

- Building the vignette PDFs in packages `grid` and `utils` is now part of running `make` from an SVN checkout on a Unix-alike: a separate `make vignettes` step is no longer required.

These vignettes are now made with `keep.source = TRUE` and hence will be laid out differently.

- `make install-strip` failed under some configuration options.
- Packages can customize non-standard installation of compiled code via a `src/install.libs.R` script. This allows packages that have architecture-specific binaries (beyond the package's shared objects/DLLs) to be installed in a multi-architecture setting.

SWEAVE & VIGNETTES

- `Sweave()` and `Stangle()` gain an encoding argument to specify the encoding of the vignette sources if the latter do not contain a '`\usepackage[]\{inputenc}`' statement specifying a single input encoding.
- There is a new `Sweave` option `figs.only = TRUE` to run each figure chunk only for each selected graphics device, and not first using the default graphics device. This will become the default in R 2.14.0.
- `Sweave` custom graphics devices can have a custom function `foo.off()` to shut them down.
- Warnings are issued when non-portable file-names are found for graphics files (and chunks if `split = TRUE`). Portable names are regarded as alphanumeric plus hyphen, underscore, plus and hash (periods cause problems with recognizing file extensions).
- The `Rtangle()` driver has a new option `show.line.nos` which is by default false; if true it annotates code chunks with a comment giving the line number of the first line in the sources (the behaviour of R $\geq 2.12.0$).
- Package installation tangles the vignette sources: this step now converts the vignette sources from the vignette/package encoding to the current encoding, and records the encoding (if not ASCII) in a comment line at the top of the installed '`.R`' file.

LICENCE

- No parts of R are now licensed solely under GPL-2. The licences for packages `rpart` and `survival` have been changed, which means that the licence terms for R as distributed are GPL-2 | GPL-3.

DEPRECATED AND DEFUNCT

- The internal functions `.readRDS()` and `.saveRDS()` are now deprecated in favour of the public functions `readRDS()` and `saveRDS()` introduced in R 2.13.0.
- Switching off lazy-loading of code via the 'LazyLoad' field of the 'DESCRIPTION' file is now deprecated. In future all packages will be lazy-loaded.
- The off-line `help()` types '"postscript"' and '"ps"' are deprecated.

UTILITIES

- R CMD check on a multi-architecture installation now skips the user's '`Renviron`' file for the architecture-specific tests (which do read the architecture-specific '`Renviron.site`' files). This is consistent with single-architecture checks, which use '`--no-environ`'.
- R CMD build now looks for 'DESCRIPTION' fields 'BuildResaveData' and 'BuildKeepEmpty' for per-package overrides. See 'Writing R Extensions'.

BUG FIXES

- `plot.lm(which = 5)` was intended to order factor levels in increasing order of mean standardized residual. It ordered the factor labels correctly, but could plot the wrong group of residuals against the label. (PR#14545)
- `mosaicplot()` could clip the factor labels, and could overlap them with the cells if a non-default value of `cex.axis` was used. (Related to PR#14550.)
- `dataframe[[row,col]]` now dispatches on `[[` methods for the selected column. (Spotted by Bill Dunlap).
- `sort.int()` would strip the class of an object, but leave its object bit set. (Reported by Bill Dunlap.)

- `pbirthday()` and `qbirthday()` did not implement the algorithm exactly as given in their reference and so were unnecessarily inaccurate. `pbirthday()` now solves the approximate formula analytically rather than using `uniroot()` on a discontinuous function.

The description of the problem was inaccurate: the probability is a tail probability ('*2 or more* people share a birthday')

- Complex arithmetic sometimes warned incorrectly about producing NAs when there were NaNs in the input.
- `seek(origin = "current")` incorrectly reported it was not implemented for a `gzfile()` connection.
- `c()`, `unlist()`, `cbind()` and `rbind()` could silently overflow the maximum vector length and cause a segfault. (PR#14571)
- The `fonts` argument to `X11(type = "Xlib")` was being ignored.
- Reading (e.g. with `readBin()`) from a raw connection was not advancing the pointer, so successive reads would read the same value. (Spotted by Bill Dunlap.)
- Parsed text containing embedded newlines was printed incorrectly by `as.character.srcref()`. (Reported by Hadley Wickham.)
- `decompose()` used with a series of a non-integer number of periods returned a seasonal component shorter than the original series. (Reported by Rob Hyndman.)

- `fields = list()` failed for `setRefClass()`. (Reported by Michael Lawrence.)
- Reference classes could not redefine an inherited field which had class "ANY". (Reported by Janko Thyson.)
- Methods that override previously loaded versions will now be installed and called. (Reported by Iago Mosqueira.)
- `addmargins()` called `numeric(apos)` rather than `numeric(length(apos))`.
- The HTML help search sometimes produced bad links. (PR#14608)
- Command completion will no longer be broken if `tail.default()` is redefined by the user. (Problem reported by Henrik Bengtsson.)
- LaTeX rendering of markup in titles of help pages has been improved; in particular, `\eqn{}` may be used there.
- `isClass()` used its own namespace as the default of the `where` argument inadvertently.
- Rd conversion to latex mis-handled multi-line titles (including cases where there was a blank line in the '`\title`' section). (It seems this happened only in 2.13.0 patched.)
- `postscript()` with an sRGB colormodel now uses sRGB for raster images (in R 2.13.[01] it used uncalibrated RGB).

There is no longer an undocumented 21845-pixel limit on raster images.

Changes on CRAN

2011-06-01 to 2011-11-30

by Kurt Hornik and Achim Zeileis

New packages in CRAN task views

Bayesian BCBCSF, PottsUtils, bayesQR, profdpm.

ChemPhys hyperSpec.

Cluster mrite, protoclust, tclust.

Distributions AtelieR, glogis, smoothmest.

Econometrics mvProbit, rms.

Environmetrics EcoHydRology, HydroMe, aqp, climatol, forecast, hydroGOF, hydroTSM, latticeDensity, oce, openair, seas, simba, soil.spec, soiltexture, topmodel, wasim.

ExperimentalDesign DiceEval, DiceView, GAD, TEQR, Vdgraph, asd, gsDesign, mkssd, mxkssd, odprism, osDesign, qtDesign, qualityTools, support.CEs.

Finance NMOF, PairTrading, betategarch, lifecontingencies, maRketSim, opefimor, rugarch.

HighPerformanceComputing OpenCL.

NaturalLanguageProcessing RTextTools.

OfficialStatistics SeleMix, deducorrect, editrules.

Optimization ROI*, Rmosek, minpack.lm.

Phylogenetics OUwie, caper, distory, motmot, phyllosim, phytools, treebase.

Psychometrics EstCRM, KernSmoothIRT, TripleR, cacIRT, fwdmsa, mRm, mirt*, mixRasch, modelfree, psychomix, psychotools.

Robust FRB, RobAStBase, RobustAFT, multinomRob, rgam.

SocialSciences rms.

Spatial Metadata, UScensus2000blkgrp, UScensus2000cdp, UScensus2000tract, adehabitat, landsat.

TimeSeries carma, cts, hydroGOF, hydroTSM, wq.

(* = core package)

New contributed packages

ACNE, ANN, APSIMBatch, AUCRF, AWS.tools, Aelasticnet, AtelieR, AutoSEARCH, BCBCSF, BN-Pdensity, BaSTA, Barnard, BinNor, BiomarkeR, CDM, CHCN, CLSOCP, CONOR, CONORData, CRM, CatDyn, CityPlot, CommonTrend, CompModSA, DATforDCEMRI, DPM.GGM, DWD, DiagTest3Grp, DirichletReg, Dodge, EL, EcoHydRology, EstCRM, ExomeCNV, FFD, FIAR, FLLat, FastRWeb, FlexParamCurve, ForImp, FourScores, GGEBiplotGUI, GMD, GMMBoost, GUTS, GWA-toolbox, GhcnDaily, HAC, HMP, HMPTrees, HTSCluster, HiveR, Hotelling, IBDsim, ISDA.R, ISIPTA, ISOweek, JoSAE, KRLS, KernSmoothIRT, KoNLP, LDcorSV, LEAPFrOG, LGS, LN3GV, LPDvar, LaF, Lahman, LargeRegression, LatticeKrig, LifeTables, M3, MAINT.Data, MBCluster.Seq, MCUSUM, MDM, MFSAS, MIPHENO, MPCl, MPTinR, MSeasy, MSeasyTkGUI, MVPARTwrap, MVR, Maeswrap, Mangrove, MatrixEQTL, McParre, MergeGUI, Metadata, MigClim, MindOnStats, MissingDataGUI, MultiPhen, NBDdirichlet, NMOF, NPsimex, NSA, NanoStringNorm, NeMo, ODB, ORCME, OUwie, OpenCL, OpenRepGrid, PACE, PAWL, PCICt, PP, PSCBS, PairTrading, PerfMeas, PhaseType, PoiClaClu, PopGenKit, PrecipStat, QLSpline, R1magic, R2STATS, RDF, RE-GENT, RHT, RHive, RMAWGEN, RMongo, RNetLogo, ROI, ROI.plugin.glpk, ROI.plugin.quadprog, ROI.plugin.symphony, RRF, RSKC, RSofia, RTextTools, RVowpalWabbit, RWeather, RXKCD, Rcell, Rclusterpp, RcmdrPlugin.KMggplot2, RcmdrPlugin.coin, RcmdrPlugin.mosaic, RcppEigen, Rdrools, Rdroolsjars, Rearrangement, Resource-Selection, Rfun, RghcnV3, Rjms, Rjmsjars, Rknots, Rlof, Rmosek, RobustAFT, Rwinsteps, SBSA, SNPRelate, SPEI, SPIAssay, SVGMapping, SearchTrees, SeleMix, SpatialVx, SpatioTemporal, SportsAnalytics, StateTrace, StochaTR, StreamingLm, SuperLearner, Survgini, SynergizeR, TAHMMAannot, TDMR, TSPC, TextRegression, ThreeWay, Tinflex, TradeStrategyAnalyzer, Tsphere, UScensus2010, VBLPCM, VBMA4hmm, Vdgraph, VisCov, WaveletCo, abcdeFBA, abn, agrisk, allelematch, alphashape3d, anaglyph, aqfig, arf3DS4, arrayhelpers, ascrda, auteur, bamdit, barcode, batade, bcool, bdpy, betategarch, betfairly, bfa, bimetallic, binomTools, bios2mds, biseVec, blm, blme, bststats, bwsurvival, cacIRT, calibFit, calmate, caper, capushe, cardidates, caribou, cccrm, cda, cghseg, cit, climdex.pcic, clpAPI, clusterPower, coefplot, colorout, comorbidities, confReg, conjoint, cotrend, cplexAPI, cplm, crimCV, crn, crs, carma, cvTools, cvplogistic, cyphid, cytoDiv, dbstats, dcens, dcmle, detect, devtools, dgof, dhglm,

dielectric, **dinamic**, **dmRTools**, **doRNG**, **drawExpression**, **dynpred**, **dynsurv**, **ebal**, **edcc**, **edesign**, **eigeninv**, **elec.strat**, **emdistr**, **emma**, **epade**, **etable**, **evtree**, **excel.link**, **ezsim**, **fairselect**, **fdaMixed**, **fdrci**, **features**, **finebalance**, **fitDRC**, **flexsurv**, **flip**, **forams**, **fpp**, **fun**, **gRim**, **gamboostLSS**, **gb**, **gbRd**, **gcmr**, **genridge**, **ggdendro**, **ggmap**, **glm2**, **glmmLasso**, **glogis**, **glpkAPI**, **goric**, **gpairs**, **anovaGG**, **graphComp**, **gridSVG**, **growthrate**, **gsmoothr**, **gstudio**, **gtx**, **gwrr**, **h5r**, **hdml**, **holdem**, **iFad**, **iRefR**, **iRegression**, **idr**, **informR**, **int64**, **intRegGOF**, **intergraph**, **isopat**, **iteRates**, **itsmr**, **jpeg**, **kBestShortestPaths**, **kinship2**, **klausuR**, **last.call**, **latticeDensity**, **lazyWeave**, **lgcp**, **libamtrack**, **lifecontingencies**, **liso**, **lmmfit**, **logconcens**, **logcondiscr**, **lorec**, **maMarketSim**, **mail**, **maxent**, **metaLik**, **metrumrg**, **minimax**, **mirt**, **mixcat**, **mixexp**, **mleur**, **mmeln**, **motmot**, **mpc**, **multicool**, **multitable**, **mvProbit**, **mvt-splot**, **ndl**, **ndvits**, **nfda**, **nloptr**, **objectProperties**, **objectSignals**, **obliqueRF**, **ocomposition**, **odprism**, **opefimor**, **openCPU.encode**, **oposSOM**, **orsk**, **osDesign**, **osmar**, **paloma**, **partykit**, **pblktest**, **pcrcoal**, **pdist**, **permute**, **phytools**, **pks**, **planar**, **playitbyr**, **plsRbeta**, **plumbr**, **polyphemus**, **polytomous**, **ppcor**, **ppiPre**, **prLogistic**, **pragma**, **probemapper**, **productplots**, **protoclust**, **psychomix**, **psychotools**, **pxR**, **qPCR.CT**, **qmrmrparser**, **qqplotter**, **qrfactor**, **qtlmt**, **rJPSGCS**, **rJava**, **randomNames**, **rasclass**, **rasterVis**, **rationalfun**, **rdatamarket**, **recommenderlabBX**, **recommenderlabJester**, **reglogit**, **rfPermute**, **rfishbase**, **rje**, **rmongodb**, **robeth**, **robustreg**, **rockchalk**, **rococo**, **roxygen2**, **rplotengine**, **rrBlupMethod6**, **rrlda**, **rsae**, **rsem**, **rtf**, **rtfbs**, **rugarch**, **rzmq**, **s4vd**, **saemix**, **sampSurf**, **sas7bdat**, **scales**, **semdiag**, **separationplot**, **siatclust**, **smcUtils**, **smfsb**, **softclassval**, **sperich**, **spider**, **spt**, **squash**, **sss**, **standGL**, **stratasphere**, **stremo**, **support.CEs**, **survC1**, **swst**, **taRifx**, **tables**, **tabplotGTK**, **tabuSearch**, **tilting**, **topolo-**

gyGSA, **treebase**, **trifield**, **truncSP**, **txtplot**, **useful**, **vegclust**, **viopoints**, **walkscoreAPI**, **weightedScores**, **whisker**, **wordcloud**, **xgrid**

Other changes

- The following packages which were previously double-hosted on CRAN and Bioconductor were moved to the Archive, and are now solely available from Bioconductor: **CORREP**, **Icens**, **RBGL**, **RLMM**, **gpls**, **graphite**, **hopach**, **impute**, **maanova**, **qvalue**.
- The following packages were moved to the Archive: **ADaCGH**, **Ardec**, **Bhat**, **DescribeDisplay**, **Design**, **LLdecomp**, **MAMA**, **MHadaptive**, **Peak2Trough**, **QRMLib**, **RBrownie**, **RStackExchange**, **Rsac**, **SQLiteDF**, **TreeRank**, **ZIGP**, **arrayImpute**, **arrayMissPattern**, **atmi**, **bdoc**, **bs**, **diseasemapping**, **fArma**, **fCalendar**, **fSeries**, **fUtilities**, **flexCrossHaz**, **googlebigquery**, **halp**, **infochimps**, **irtProb**, **maticce**, **miniGUI**, **muS2RC**, **muStat**, **muUtil**, **npmc**, **powerpkg**, **tpsDesign**, **uniPlot**.
- The following packages were resurrected from the Archive: **SNPMaP**, **SNPMaP.cdm**, **SoPhy**, **birch**, **csampling**, **cts**, **elec**, **formula.tools**, **hi erfstat**, **marg**, **sabreR**.

Kurt Hornik

WU Wirtschaftsuniversität Wien, Austria

Kurt.Hornik@R-project.org

Achim Zeileis

Universität Innsbruck, Austria

Achim.Zeileis@R-project.org

News from the Bioconductor Project

We are pleased to announce Bioconductor 2.9, released on 14 October 2011. Bioconductor 2.9 is compatible with R 2.14.0, and consists of 516 software packages and more than 500 up-to-date annotation packages. There are 60 new software packages, and enhancements to many others. Explore Bioconductor at <http://bioconductor.org>. Install packages with

```
> source("http://bioconductor.org/biocLite.R")
> # install standard packages...
> biocLite()
> # ...or only VariantAnnotation
> biocLite("VariantAnnotation")
```

A Bioconductor Amazon Machine Instance is available; see <http://bioconductor.org/help/bioconductor-cloud-ami>.

New and revised packages

This release includes new packages for diverse areas of high-throughput analysis. Highlights include:

High-throughput sequencing: **DEXSeq**, **DiffBind**, **EDASeq**, **REDseq**, **ReQON**, **Repitoools**, **TSSi**, **VariantAnnotation**, **cummeRbund**, **fastseg**, **htSeqTools**, **nucleR**, **r3Cseq**, **tweeDEseq**.

Microarrays: **AGDEX**, **CNAnorm**, **CellNOptR**, **Cormotif**, **DECIPHER**, **GWASTools**, **Mm-PalateMiRNA**, **cn.mops**, **cqn**, **exomeCopy**, **minfi**, **rqubic**, **stepwiseCM**.

Advanced statistical implementations: **DTA**, **RCAS-PAR**, **dks**, **randPack**, **sva**, **trigger**.

Networks and pathways: **DOSE**, **GOFunction**, **grenits**, **GeneExpressionSignature**, **IdMappingRetrieval**, **PAN**, **PREDA**, **RTopper**, **RamiGO**, **RedeR**, **graphite**, **inSilicoDb**, **predictionet**.

Mass spectrometry, qPCR, and flow cytometry: **NormqPCR**, **ReadqPCR**, **flowType**, **flowWorkspace**, **iontree**, **isobar**, **mzR**, **ncdfFlow**.

Visualization: **biovizBase**, **ggbio**, **qtbase**, **qtpaint**.

Our large collection of microarray- and organism-specific annotation packages have, as usual, been updated to include current information. This release introduces a select interface to annotation packages, simplifying queries for multiple types of annotation

within and between data sources. The new **VariantAnnotation** package processes VCF files, including interfaces to SIFT and PolyPhen data bases for assessing consequences of sequence changes.

Further information on new and existing packages can be found on the Bioconductor web site; 'BiocViews' identify coherent groups of packages, with links to package descriptions, vignettes, reference manuals, and use statistics. Our web site has been enhanced with easier access to package NEWS and other support files.

Other activities

A meeting in December highlighted contributions from our European developer community; we look forward to Seattle's Annual Meeting on 23-25 July 2012. The active Bioconductor mailing lists (<http://bioconductor.org/help/mailing-list/>) connect users with each other, to domain experts, and to maintainers eager to ensure that their packages satisfy the needs of leading edge approaches. Bioconductor package maintainers and the Bioconductor team invest considerable effort in producing high-quality software. The Bioconductor team continues to ensure quality software through technical and scientific reviews of new packages, and daily builds of released packages on Linux, Windows, and MacOS platforms.

Looking forward

Our contributors provide a tremendous diversity of high-quality packages. These are enhancing areas of existing strength in statistical analysis of microarrays, while driving project growth in high-throughput sequence analysis and emerging domains like mass spectrometry and qPCR. These directions provide an important incentive for infrastructure, contributed by the Bioconductor core team and others, to support processing and visualization of very large data sets.

Important directions for the Bioconductor core team include representation and manipulation of complex sequence alignments and variants, convenient and integrated approaches to gene and genome annotation, continued efforts to ease cloud-based use of Bioconductor, and approaches to ease development and support of increasingly inter-related packages.

*Bioconductor Team
Program in Computational Biology
Fred Hutchinson Cancer Research Center*

R Foundation News

by Kurt Hornik

Donations and new members

Donations

Geoffrey S Hubona, The Georgia R School, USA
Packt Publishing Limited, UK

New supporting members

Philippe Bartil Lecavalier, Canada
Bernhard Lehnert, Germany
Joshua Wiley, USA
Thomas Zumbrunn, Switzerland

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