News

The Newsletter of the R Project Volume 8/2, October 2008

Editorial

by John Fox

I had the opportunity recently to examine the development of the R Project. A dramatic index of this development is the rapid growth in R packages, which is displayed in Figure 1; because the vertical axis in this graph is on a log scale, the least-squares linear-regression line represents exponential growth.

This issue of R News is another kind of testament to the richness and diversity of applications programmed in R: Hadley Wickham, Michael Lawrence, Duncan Temple Lang, and Deborah Swayne describe the **rggobi** package, which provides an interface to GGobi, software for dynamic and interactive statistical graphics. Carsten Dormann, Bernd Gruber, and Jochen Fründ introduce the **bipartite** package for examining interactions among species in ecological communities. Ionnis Kosmidis explains how to use the **profileModel** package to explore likelihood surfaces and construct confidence intervals for parameters of models fit by maximum-likelihood and closely related methods. Ingo Feinerer illustrates how the **tm** ("text-mining") package is employed for the analysis of textual data. Yihui Xie and Xiaoyne Cheng demonstrate the construction of statistical animations in R with the **animation** package. Thomas Yee introduces the **VGAM** package, which is capable of fitting a dizzying array of statistical models. Paul Murrell shows how the **compare** package is used to locate similarities and differences among non-identical objects, and explains how the package may be applied to grading students' assignments. Arthur Allignol, Jan Beyersmann, and Martin Schumacher describe the **mvna** package for the Nelson-Aalen estimator of cumulative transition hazards in multistate models. Finally, in the latest installment of the *Programmers' Niche* column, Vince Carey reveals the mind-bending qualities of recursive computation in R without named self-reference.

The "news" portion of this issue of *R Rnews* includes a report on the recent *useR!* conference in Dortmund, Germany; announcements of *useR! 2009* in Rennes, France, and of *DSC 2009* in Copenhagen, Denmark, both to be held in July; news from the R Foundation and from the Bioconductor Project; the latest additions to CRAN; and changes in the recently released version 2.8.0 of R.

Unless we accumulate enough completed articles to justify a third issue in 2008, this will be my last issue as editor-in-chief of *R News*. Barring a third issue in 2008, this is also the last issue of *R News* — but don't panic! *R News* will be reborn in 2009 as *The R Journal*. Moreover, the purpose and general character of the publication will be largely unchanged. We

Contents of this issue:

think that the new name better reflects the direction in which *R News* has evolved, and in particular the quality of the refereed articles that we publish.

During my tenure as an *R News* editor, I've had the pleasure of working with Vince Carey, Peter Dalgaard, Torsten Hothorn, and Paul Murrell. I look forward to working next year with Heather Turner, who is joining the editorial board in 2009, which will be my last year on the board.

John Fox
McMaster University
Canada
John.Fox@R-project.org

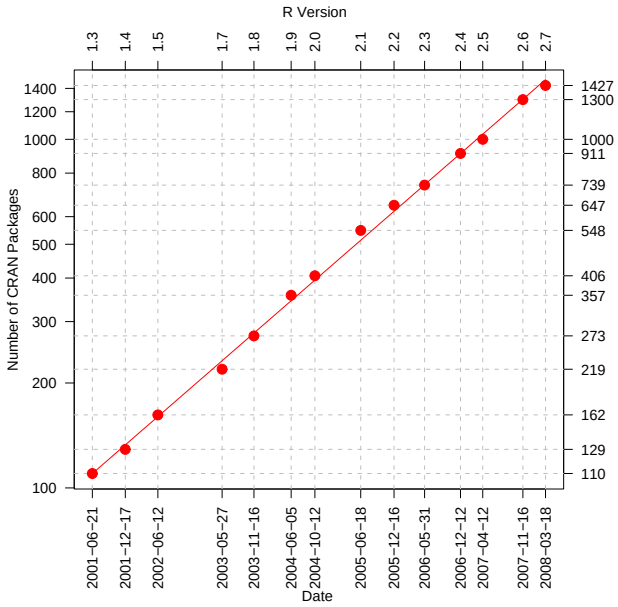


Figure 1: The number of R packages on CRAN has grown exponentially since R version 1.3 in 2001. *Source of Data*: https://svn.r-project.org/R/branches/.

An Introduction to rggobi

Hadley Wickham, Michael Lawrence,
Duncan Temple Lang, Deborah F Swayne

Introduction

The rggobi (Temple Lang and Swayne, 2001) package provides a command-line interface to GGobi, an interactive and dynamic graphics package. Rggobi complements GGobi's graphical user interface by enabling fluid transitions between analysis and exploration and by automating common tasks. It builds on the first version of rggobi to provide a more robust and user-friendly interface. In this article, we show how to use ggobi and offer some examples of the insights that can be gained by using a combination of analysis and visualisation.

This article assumes some familiarity with GGobi. A great deal of documentation, both introductory and advanced, is available on the GGobi web site, `http://www.ggobi.org`; newcomers to GGobi might find the demos at `ggobi.org/docs` especially helpful. The software is there as well, both source and executables for several platforms. Once you have installed GGobi, you can install rggobi and its dependencies using `install.packages("rggobi", dep=T)`.

This article introduces the three main components of rggobi, with examples of their use in common tasks:

- getting data into and out of GGobi;
- modifying observation-level attributes ("automatic brushing");
- basic plot control.

We will also discuss some advanced techniques such as creating animations with GGobi, the use of edges, and analysing longitudinal data. Finally, a case study shows how to use rggobi to create a visualisation for a statistical algorithm: manova.

Data

Getting data from R into GGobi is easy: `g <- ggobi(mtcars)`. This creates a `GGobi` object called `g`. Getting data out isn't much harder: Just index that `GGobi` object by position (`g[[1]]`) or by name (`g[["mtcars"]]` or `g$mtcars`). These return `GGobiData` objects which are linked to the data in GGobi. They act just like regular data frames, except that changes are synchronised with the data in the corresponding GGobi. You can get a static copy of the data using `as.data.frame`.

Once you have your data in GGobi, it's easy to do something that was hard before: find multivariate outliers. It is customary to look at uni- or bivariate plots to look for uni- or bivariate outliers, but higher-dimensional outliers may go unnoticed. Looking for these outliers is easy to do with the tour. Open your data with GGobi, change to the tour view, and select all the variables. Watch the tour and look for points that are far away or move differently from the others—these are outliers.

Adding more data sets to an open GGobi is also easy: `g$mtcars2 <- mtcars` will add another data set named "mtcars2". You can load any file type that GGobi recognises by passing the path to that file. In conjunction with `ggobi_find_file`, which locates files in the GGobi installation directory, this makes it easy to load GGobi sample data. This example loads the olive oils data set included with GGobi:

```
library(rggobi)
ggobi(ggobi_find_file("data", "olive.csv"))
```

Modifying observation-level attributes, or "automatic brushing"

Brushing is typically thought of as an operation performed in a graphical user interface. In GGobi, it refers to changing the colour and symbol (glyph type and size) of points. It is typically performed in a linked environment, in which changes propagate to every plot in which the brushed observations are displayed. in GGobi, brushing includes shadowing, where points sit in the background and have less visual impact, and exclusion, where points are completed excluded from the plot. Using rggobi, brushing can be performed from the command line; we think of this as "automatic brushing." The following functions are available:

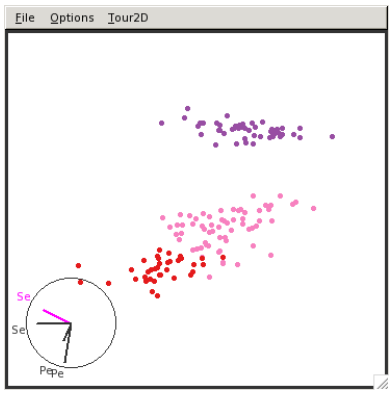
- change glyph colour with `glyph_colour` (or `glyph_color`)
- change glyph size with `glyph_size`
- change glyph type with `glyph_type`
- shadow and unshadow points with `shadowed`
- exclude and include points with `excluded`

Each of these functions can be used to get or set the current values for the specified `GGobiData`. The "getters" are useful for retrieving information that you have created while brushing in GGobi, and the "setters" can be used to change the appearance of points based on model information, or to create animations. They can also be used to store, and then later recreate, the results of a complicated sequence of brushing steps.

This example demonstrates the use of `glyph_colour` to show the results of clustering the

infamous Iris data using hierachical clustering. Using GGobi allows us to investigate the clustering in the original dimensions of the data. The graphic shows a single projection from the grand tour.

```
g <- ggobi(iris)
clustering <- hclust(dist(iris[,1:4]),
  method="average")
glyph_colour(g[1]) <- cuttree(clustering, 3)
```



Another function, `selected`, returns a logical vector indicating whether each point is currently enclosed by the brush. This could be used to further explore interesting or unusual points.

Displays

A `GGobiDisplay` represents a window containing one or more related plots. With rggobi you can create new displays, change the projection of an existing plot, set the mode which determines the interactions available in a display, or select a different set of variables to plot.

To retrieve a list of displays, use the `displays` function. To create a new display, use the `display` method of a `GGobiData` object. You can specify the plot type (the default is a bivariate scatterplot, called "XY Plot") and variables to include. For example:

```
g <- ggobi(mtcars)
display(g[1], vars=list(X=4, Y=5))
display(g[1], vars=list(X="drat", Y="hp"))
display(g[1], "Parallel Coordinates Display")
display(g[1], "2D Tour")
```

The following display types are available in GGobi (all are described in the manual, available from `ggobi.org/docs`):

Name	Variables
1D Plot	1 X
XY Plot	1 X, 1 Y
1D Tour	*n* X
Rotation	1 X, 1 Y, 1 Z
2D Tour	*n* X
2x1D Tour	*n* X, *n* Y
Scatterplot Matrix	*n* X
Parallel Coordinates Display	*n* X
Time Series	1 X, *n* Y
Barchart	1 X

After creating a plot you can get and set the displayed variables using the `variable` and `variable<-` methods. Because of the range of plot types in GGobi, variables should be specified as a list of one or more named vectors. All displays require an `X` vector, and some require `Y` and even `Z` vectors, as specified in the above table.

```
g <- ggobi(mtcars)
d <- display(g[1],
  "Parallel Coordinates Display")
variables(d)
variables(d) <- list(X=8:6)
variables(d) <- list(X=8:1)
variables(d)
```

A function which saves the contents of a GGobi display to a file on disk, is called `ggobi_display_save_picture`. This is what we used to create the images in this document. This creates an exact (raster) copy of the GGobi display. If you want to create publication quality graphics from GGobi, have a look at the `DescribeDisplay` plugin and package at `http://www.ggobi.org/describe-display`. These create R versions of GGobi plots.

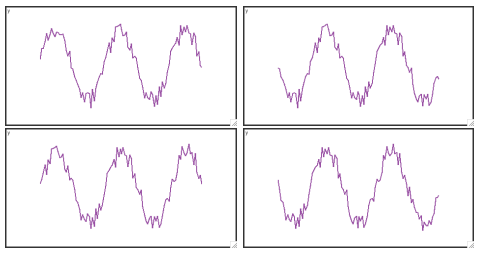
To support the construction of custom interactive graphics applications, rggobi enables the embedding of GGobi displays in graphical user interfaces (GUIs) based on the RGtk2 package. If `embed = TRUE` is passed to the `display` method, the display is not immediately shown on the screen but is returned to R as a `GtkWidget` object suitable for use with RGtk2. Multiple displays of different types may be combined with other widgets to form a cohesive GUI designed for a particular data analysis task.

Animation

Any changes that you make to the `GGobiData` objects are updated in GGobi immediately, so you can easily create animations. This example scrolls through a long time series:

```
df <- data.frame(
  x=1:2000,
  y=sin(1:2000 * pi/20) + runif(2000, max=0.5)
)
```

```
g <- ggobi_longitudinal(df[1:100, ])

df_g <- g[1]
for(i in 1:1901) {
  df_g[, 2] <- df[i:(i + 99), 2]
}
```



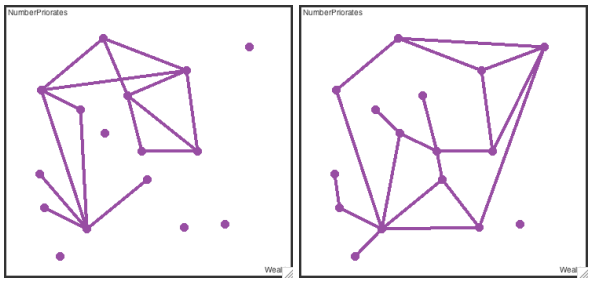
Edge data

In GGobi, an edge data set is treated as a special type of dataset in which a record describes an edge – which may still be associated with an n-tuple of data. They can be used to represent many differentent types of data, such as distances between observations, social relationships, or biological pathways.

In this example we explore marital and business relationships between Florentine families in the 15th century. The data comes from the `ergm` (social networking analysis) package (Handcock et al., 2003), in the format provided by the `network` package (Butts et al., July 26, 2008).

```
install.packages(c("network", "ergm"))
library(ergm)
data(florentine)

flo <- data.frame(
  priorates = get.vertex.attribute(
    flobusiness, "priorates"
  ),
  wealth = get.vertex.attribute(
    flobusiness, "wealth"
  )
)

families <- network.vertex.names(flobusiness)
rownames(flo) <- families

edge_data <- function(net) {
  edges <- as.matrix.network(
    net,
    matrix.type="edgelist"
  )
  matrix(families[edges], ncol=2)
}

g <- ggobi(flo)
edges(g) <- edge_data(flomarriage)
edges(g) <- edge_data(flobusiness)
```

This example has two sets of edges because some pairs of families have marital relationships but not business relationships, and vice versa. We can use the edges menu in GGobi to change between the two edge sets and compare the relationship patterns they reveal.



How is this stored in GGobi? An edge dataset records the names of the source and destination observations for each edge. You can convert a regular dataset into an edge dataset with the `edges` function. This takes a matrix with two columns, source and destination names, with a row for each edge observation. Typically, you will need to add a new data frame with number of rows equal to the number of edges you want to add.

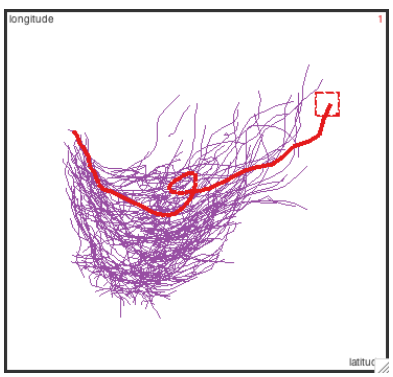
Longitudinal data

A special case of data with edges is time series or longitudinal data, in which observations adjacent in time are connected with a line. Rggobi provides a convenient function for creating edge sets for longitudinal data, `ggobi_longitudinal`, that links observations in sequential time order.

This example uses the `stormtracks` data included in rggobi. The first argument is the dataset, the second is the variable specifying the time component, and the third is the variable that distinguishes the observations.

```
ggobi_longitudinal(stormtracks, seasday, id)
```

For regular time series data (already in order, with no grouping variables), just use `ggobi_longitudinal` with no other arguments.



Case study

This case study explores using rggobi to add model information to data; here will add confidence ellipsoids around the means so we can perform a graphical manova.

The first (and most complicated) step is to generate the confidence ellipsoids. The `ellipse` function does this. First we generate random points on the surface of sphere by drawing *npoints* from a random normal distribution and standardising each dimension. This sphere is then skewed to match the desired variance-covariance matrix, and its is size adjusted to give the appropriate *cl*-level confidence ellipsoid. Finally, the ellipsoid is translated to match the column locations.

```
conf.ellipse <- function(data, npoints=1000,
  cl=0.95, mean=colMeans(data), cov=var(data),
  n=nrow(data)
) {
  norm.vec  <-  function(x) x / sqrt(sum(x^2))

  p <- length(mean)
  ev <- eigen(cov)

  normsamp <- matrix(rnorm(npoints*p), ncol=p)
  sphere <- t(apply(normsamp, 1, norm.vec))

  ellipse <- sphere %*%
    diag(sqrt(ev$values)) %*% t(ev$vectors)
  conf.region <- ellipse * sqrt(p * (n-1) *
    qf(cl, p, n-p) / (n * (n-p)))
  if (!missing(data))
    colnames(ellipse) <- colnames(data)

  conf.region + rep(mean, each=npoints)
}
```

This function can be called with a data matrix, or with the sufficient statistics (mean, covariance matrix, and number of points). We can look at the output with ggobi:

```
ggobi(conf.ellipse(
  mean=c(0,0), cov=diag(2), n = 100))

cv <- matrix(c(1,0.15,0.25,1),
  ncol=2, n = 100)
ggobi(conf.ellipse(
  mean=c(1,2), cov=cv, n = 100))

mean <- c(0,0,1,2)
ggobi(conf.ellipse(
  mean=mean, cov=diag(4), n = 100))
```

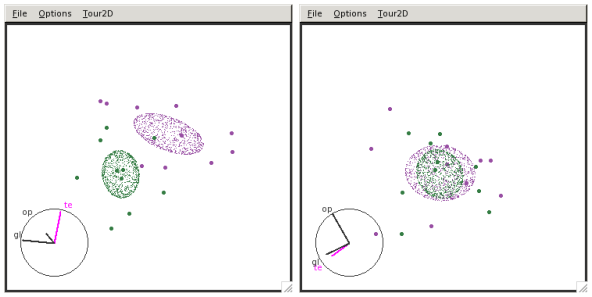
In the next step, we will need to take the original data and supplement it with the generated ellipsoid:

```
manovaci <- function(data, cl=0.95) {
  dm <- data.matrix(data)
  ellipse <- as.data.frame(
    conf.ellipse(dm, n=1000, cl=cl)
  )

  both <- rbind(data, ellipse)
  both$SIM <- factor(
    rep(c(FALSE, TRUE), c(nrow(data), 1000))
  )

  both
}
ggobi(manovaci(matrix(rnorm(30), ncol=3)))
```

Finally, we create a method to break a dataset into groups based on a categorical variable and compute the mean confidence ellipsoid for each group. We then use the automatic brushing functions to make the ellipsoid distinct and to paint each group a different colour. Here we use 68% confidence ellipsoids so that non-overlapping ellipsoids are likely to have significantly different means.

```
ggobi_manova <- function(data, catvar,
    cl=0.68) {
  each <- split(data, catvar)
  cis <- lapply(each, manovaci, cl=cl)

  df <- as.data.frame(do.call(rbind, cis))
  df$var <- factor(rep(
    names(cis), sapply(cis, nrow)
  ))

  g <- ggobi(df)
  glyph_type(g[1]) <- c(6,1)[df$SIM]
  glyph_colour(g[1]) <- df$var
  invisible(g)
}
```

These images show a graphical manova. You can see that in some projections the means overlap, but in others they do not.



Conclusion

GGobi is designed for data exploration, and its integration with R through rggobi allows a seamless workflow between analysis and exploration. Much of the potential of rggobi has yet to be realized,

but some ideas are demonstrated in the `classifly` package (Wickham, 2007), which visualises high-dimensional classification boundaries. We are also keen to hear about your work—if you develop a package using rggobi please let us know so we can highlight your work on the GGobi homepage.

We are currently working on the infrastructure behind GGobi and rggobi to allow greater control from within R. The next generation of rggobi will offer a direct low-level binding to the public interface of every GGobi module. This will coexist with the high-level interface presented in this paper. We are also working on consistently generating events in GGobi so that you will be able respond to events of interest from your R code. Together with the RGtk2 package (Lawrence and Temple Lang, 2007), this should allow the development of custom interactive graphics applications for specific tasks, written purely with high-level R code.

Acknowledgements

This work was supported by National Science Foundation grant DMS0706949. The ellipse example is taken, with permission, from Professor Di Cook's notes for multivariate analysis.

Bibliography

C. T. Butts, M. S. Handcock, and D. R. Hunter. *network: Classes for Relational Data*. Irvine, CA, July 26, 2008. URL `http://statnet.org/`. R package version 1.4-1.

M. S. Handcock, D. R. Hunter, C. T. Butts, S. M. Goodreau, and M. Morris. *ergm: A Package to Fit, Simulate and Diagnose Exponential-Family Models for Networks*. Seattle, WA, 2003. URL `http://CRAN.R-project.org/package=ergm`. Version 2.1 . Project home page at http://statnetproject.org.

M. Lawrence and D. Temple Lang. *RGtk2: R bindings for Gtk 2.8.0 and above*, 2007. URL `http://www.ggobi.org/rgtk2,http://www.omegahat.org`. R package version 2.12.1.

D. Temple Lang and D. F. Swayne. GGobi meets R: an extensible environment for interactive dynamic data visualization. In *Proceedings of the 2nd International Workshop on Distributed Statistical Computing*, 2001.

H. Wickham. *classifly: Explore classification models in high dimensions*, 2007. URL `http://had.co.nz/classifly`. R package version 0.2.3.

Introducing the bipartite Package: Analysing Ecological Networks

Carsten F. Dormann, Bernd Gruber and Jochen Fründ

Introduction

Interactions among species in ecological communities have long fascinated ecologists. Prominent recent examples include pollination webs (Memmott et al., 2004), species-rich predator-prey systems (Tylianakis et al., 2007) and seed dispersal mutualisms (all reviewed in Blüthgen et al., 2007). Many of the topological descriptors used in food webs since the 1960s have limited ecological meaning when only two trophic levels are investigated (for example chain length: Pimm, 1982/2002; Montoya et al., 2006). Here, the network becomes *bipartite*, i.e. every member of one trophic level is only connected to members of the other trophic level: direct interactions *within* trophic levels are regarded as unimportant. For bipartite ecological networks very many, more or less different, indices have been proposed to capture important features of the interactions and of the species. For example, species degrees (number of species the target species is linked to) or species strength (sum of level of dependencies of other species on the target) are supposed to quantify the importance of a species in a web.

The new R-package `bipartite`, introduced here, provides functions to viualise webs and calculate a series of indices commonly used to describe pattern in ecological webs. It focusses on webs consisting of only two trophic levels, e.g. pollination webs or predator-prey-webs. We had three types of ecological bipartite webs in mind when writing this package: seed-disperser, plant-pollinator and predator-prey systems.

Bipartite networks, as analysed and plotted in the package `bipartite`, can be represented by a matrix, in which, in our definition, columns represent species in the higher trophic level, and rows species in the lower trophic level. Entries in the matrix represent observed links, either quantitatively (with one to many interactions per link) or qualitatively (binary). Usually such matrices are very sparse, marginal sums (i.e. abundance distributions) highly skewed and average number of interactions per link are low (around 2: Blüthgen et al., 2007).

With the package `bipartite`, presented here, we wanted to overcome two main deficiencies in the field: 1. Lack of software to calculate various indices and topological descriptors of bipartite networks. And 2. No convenient plotting tool for bipartite networks. This article aims to briefly present the two visualisation functions (`plotweb` and `visweb`), then present an example output from the calculation of network-level descriptors (using function `networklevel`) and finally address some miscellaneous issues to do with fitting degree distributions, secondary extinction slopes and null models for bipartite webs.

Along with several functions we also include 19 data sets on pollinator networks, taken from the National Center for Ecological Analysis and Synthesis webpage devoted to this topic (`www.nceas.ucsb.edu/interactionweb`). There are several other bipartite data sets at this repository, and our data include only quantitative plant-pollinator networks.

Plotting ecological networks

The function `plotweb` draws a bipartite graph, in which rectangles represent species, and the width is proportional to the sum of interactions involving this species. Interacting species are linked by lines, whose width is again proportional to the number of interactions (but can be represented as simple lines or triangles pointing up or down). An example is given in Fig. 1 for the data set `mosquin1967`, which is included in the package and can be generated using `plotweb(mosquin1967)`.

Alternatively, the function `visweb` plots the data matrix with shading representing number of interactions per link. As default, this gives an idea about the filling of the matrix. With option `type="diagonal"`, however, `visweb` depicts compartments for easy perception (Fig.2). The same compartments are visible in Fig.1, too, due to the default sequence of species here being the arrangement used in `visweb(., type="diagonal")`. This sequence is determined, as suggested by (Lewinsohn et al., 2006), using correspondence analysis.

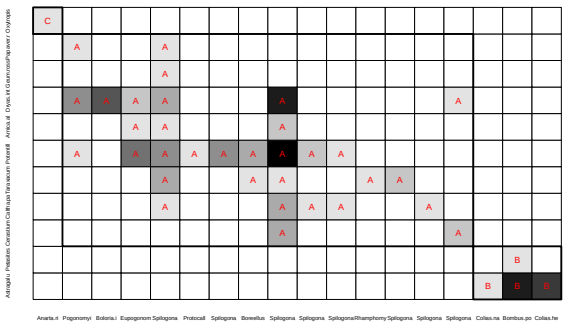


Figure 2: A plot of the network matrix produced by `visweb(mosquin1967, type="diagonal")`.

 ISSN 1609-3631

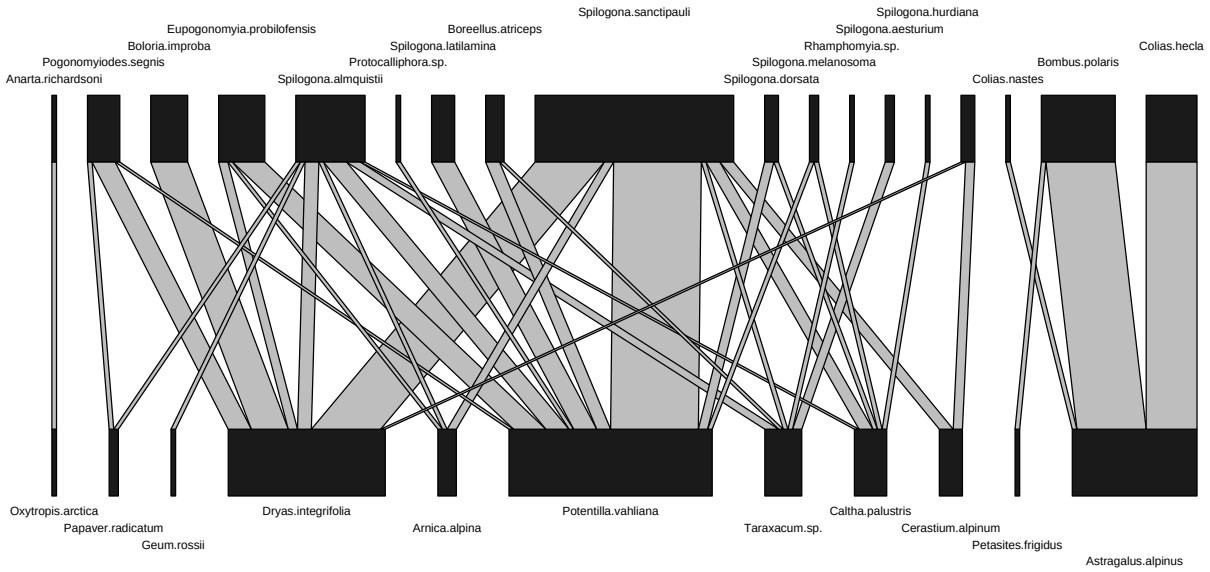


Figure 1: A bipartite graph produced by default settings of function `plotweb`.

Calculating network metrics

`bipartite` features two main functions to calculate indices: `specieslevel` and `networklevel`. The former returns various values describing, e.g., the specialisation or the dependence asymmetry of each species. Since its output is bulky, we omit it here and only present the function `networklevel` in detail, which also comprises a larger set of different indices:

```
>networklevel(mosquin1967)
$`number of higher trophic species`
[1] 18

$`number of lower trophic species`
[1] 11

$`number of links`
[1] 1.310345

$generality
[1] 2.677306

$vulnerability
[1] 4.114345

$`interaction evenness`
[1] 0.8512671

$`Alatalo interaction evenness`
[1] 0.6587369

$`number of compartments`
[1] 3

$`compartment diversity`
[1] 2.00787

$`cluster coefficient`
[1] 0.1363636

$H2
[1] 0.4964885

$`web asymmetry`
[1] 0.2413793

$`interaction strength asymmetry`
[1] 0.1607192

$`specialisation asymmetry`
[1] -0.1755229

$`extinction slope lower trophic level`
[1] 2.286152

$`extinction slope higher trophic level`
[1] 2.9211

$`degree distribution lower trophic level`
                        Estimate Std.Err Pr(>|t|)    R2   AIC
exponential             0.207527 0.02905 0.000834 0.992 -7.11
power law               0.701034 0.08856 0.000517 0.967 -8.09
trunc. power law [slope] 0.431810 0.31661 0.244321 0.987 -7.15

$`degree distribution higher trophic level`
                        Estimate Std.Err Pr(>|t|)    R2   AIC
exponential             0.221084 0.04283 0.006691 0.999 -3.21
power law               0.744383 0.12834 0.004394 0.960 -4.34
trunc. power law [slope] 0.511777 0.43347 0.322823 0.980 -2.82

$`higher trophic level niche overlap`
[1] 0.2237163

$`lower trophic level niche overlap`
[1] 0.2505869

$`mean number of shared hosts`
[1] 0.8545455

$togetherness
[1] 0.1050109

$`C-score`
[1] 0.6407096

$`V-ratio`
[1] 11.11811

$nestedness
[1] 44.28693
```

We opted for a list structure to be able to accommodate tables in the output, and because the option

`index` allows specification of the indices the user is interested in (defaults to all).

All indices are explained and/or referenced, in the help pages, so a detailed description is omitted here. Among our personal favourites are the network-wide specialisation H_2' (Blüthgen et al., 2006), generality and vulnerability (Tylianakis et al., 2007) and the small-world measure 'clustering coefficient' (Watts and Strogatz, 1998). Furthermore, we took the liberty to modify the index 'dependence asymmetry', because it has been shown to be biased (Blüthgen et al., 2007). The original formulation is available as a special case of 'interaction strength asymmetry' and can be called using `networklevel(mosquin1967, index="ISA", ISAmethod="Bascompte")`.

Miscellaneous

Three list entries may warrant particular mentioning: *Web asymmetry* is simply the ratio of matrix dimensions. In a recent paper, Blüthgen et al. (2007) showed that some indices may be particularly influenced by the matrix dimensions, and hence web asymmetry may serve as a kind of correction index. *Extinction slopes* (for lower and higher level) are hyperbolic fits to a simulated extinction sequence of the network, which causes secondary extinctions in the other trophic level (only for networks with strong dependence). The idea was presented by Memmott et al. (2004) and we include this rather rough measure as a simple implementation (see `?second.extinct` for specification of simulations and `?slope.bipartite` for details on fitting of the hyperbolic curve). Finally, *degree distributions* (for both trophic levels) have been heralded by Jordano et al. (2003) and Montoya et al. (2006) as being best described by truncated power laws, rather than exponential of simple power law functions. We fit all three, but also notice that many networks provide only 4 to 5 levels of degrees, so that a non-linear fit to so few data points gives us little confidence in its meaning, and often the fitting does not converge, due to singular gradients.

Some of the indices calculated by `networklevel` may be ecologically uninformative because they are driven by either constraints in web dimensions or are a consequence of the lognormal distribution of species abundances (e.g. nestedness). This is not to say that for specific questions these indices are not important, just to caution that in some cases statistical artefacts may confound an index' intended meaning. We can investigate this by constructing random webs, e.g. by employing Patefield's `r2dtable`-algorithm (Fig. 3).

These findings give us some hope that the observed pattern are not a mere artefact of species distributions and web dimensions. There are many different ways to construct a null model, and this is only one of them (Vázquez and Aizen, 2003). We provide two further null models, `shuffle.web` and `swap.web`. The former simply shuffles the observed values within the matrix under the constraint of retaining dimensionality; the latter contrains both marginal sums (as does `r2dtable`) *and* connectance (i.e. number of non-zero entries in the matrix). As observed connectance is much lower than connectance in random marginal-sum-constrained networks, maintaining connectance implies an ecological mechanism (such as co-evolution of pollinator and plant, body size-relationships between prey and predator, and so forth).

Although we tried to incorporate many descriptors of networks, there are certainly several missing. For example, the social literature has put much emphasis on betweenness and centrality, concepts that we find difficult to interpret in the context of `bipartite` (=two-mode) ecological networks. Some of these functions are implemented in the R-package `sna` (Social Network Analysis Butts, 2007), which can be accessed after transforming the bipartite data to one-mode graphs using `bipartite`'s function `as.one.mode`. Others can be calculated using the freeware Pajek (`http://vlado.fmf.uni-lj.si/pub/networks/pajek/`: Batagelj and Mrvar, 2003). We made, as yet, no attempt to include indices that use sophisticated optimisation algorithms (e.g. Guimerà and Amaral 2005's modularity index or Clauset et al. 2008's hierarchical structure functions), mainly because of time limitations, but also because they draw heavily on computer resources. Contributions to `bipartite` are welcomed, in order to make further progress in the issues awaiting networks in ecology (Bascompte, 2007).

Bibliography

J. Bascompte. Networks in ecology. *Basic and Applied Ecology*, 8:485–490, 2007.

V. Batagelj and A. Mrvar. Pajek - analysis and visualization of large networks. In M. Jünger and P. Mutzel, editors, *Graph Drawing Software*, pages 77–103. Springer, Berlin, 2003.

N. Blüthgen, F. Menzel, and N. Blüthgen. Measuring specialization in species interaction networks. *BMC Ecology*, 6(9):12, 2006.

N. Blüthgen, F. Menzel, T. Hovestadt, B. Fiala, and N. Blüthgen. Specialization, constraints and conflicting interests in mutualistic networks. *Current Biology*, 17:1–6, 2007.

C. T. Butts. *sna: Tools for Social Network Analysis*, 2007. URL `http://erzuli.ss.uci.edu/R.stuff`. R package version 1.5.

```
>null.t.test(mosquin1967, index=c("generality", "vulnerability", "cluster coefficient", "H2", "ISA",
+ "SA"), nrep=2, N=20)

                                     obs   null mean    lower CI    upper CI         t            P
generality                     2.6773063  4.22916358  4.15161471  4.30671245  41.88423 3.497206e-20
vulnerability                  4.1143452  7.21792806  7.07648249  7.35937363  45.92490 6.178095e-21
cluster coefficient            0.1363636  0.24545455  0.22667908  0.26423001  12.16108 2.067759e-10
H2                             0.4964885  0.14030524  0.12975531  0.15085517 -70.66398 1.800214e-24
interaction strength asymmetry 0.1607192  0.05951281  0.05190249  0.06712313 -27.83425 7.291115e-17
specialisation asymmetry      -0.1978587 -0.16827052 -0.20874785 -0.12779319   1.52996 1.425060e-01
```

Figure 3: Illustration of the `null.t.test`-function with selected network indices.

A. Clauset, C. Moore and M. E. J. Newman. Hierarchical structure and the prediction of missing links in networks. *Nature*, 453:98–101, 2008.

R. Guimerà and L. A. N. Amaral. Functional cartography of complex metabolic networks. *Nature*, 433: 895–900, 2005.

P. Jordano, J. Bascompte, and J. M. Olesen. Invariant properties in coevolutionary networks of plant-animal interactions. *Ecology Letters*, 6:69–81, 2003.

T. M. Lewinsohn, P. I. Prado, P. Jordano, J. Bascompte, and J. M. Olesen. Structure in plant-animal interaction assemblages. *Oikos*, 113(1):174–184, 2006.

J. Memmott, N. M. Waser, and M. V. Price. Tolerance of pollination networks to species extinctions. *Proceedings of the Royal Society*, 271:2605–2611, 2004.

J. M. Montoya, S. L. Pimm, and R. V. Solé. Ecological networks and their fragility. *Nature*, 442:259–264, 2006.

S. Pimm. *Food Webs*. Chicago University Press, Chicago, 1982/2002.

J. M. Tylianakis, T. Tscharntke, and O. T. Lewis. Habitat modification alters the structure of tropical host-parasitoid food webs. *Nature*, 445:202–205, 2007.

D. P. Vázquez and M. A. Aizen. Null model analyses of specialization in plant-pollinator interactions. *Ecology*, 84(9):2493–2501, 2003.

D. J. Watts and S. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

Carsten F. Dormann & Bernd Gruber
Department of Computational Landscape Ecology
UFZ Centre for Environmental Research
Permoserstr. 15
04318 Leipzig, Germany
Email: `carsten.dormann@ufz.de`

Jochen Fründ
Agroecology
University of Göttingen
Waldweg 26
37073 Göttingen
Germany

The profileModel R Package: Profiling Objectives for Models with Linear Predictors.

by Ioannis Kosmidis

Introduction

In a multi-parameter problem the profiles of the likelihood can be used for the construction of confidence intervals for parameters, as well as to assess features of the likelihood surface such as local maxima, asymptotes, etc., which can affect the performance of asymptotic procedures. The `profile` methods of the **R** language (**stats** and **MASS** packages) can be used for profiling the likelihood function for several classes of fitted objects, such as `glm` and `polr`. However, the methods are limited to cases where the profiles are almost quadratic in shape and can fail, for example, in cases where the profiles have an asymptote.

Furthermore, often the likelihood is replaced by an alternative objective for either the improvement of the properties of the estimator, or for computational efficiency when the likelihood has a complicated form (see, for example, Firth (1993) for a maximum penalized likelihood approach to bias-reduction, and Lindsay (1988) for composite likelihood methods, respectively). Alternatively, estimation might be performed by using a set of estimating equations which do not necessarily correspond to a unique objective to be optimized, as in quasi-likelihood estimation (Wedderburn, 1974; McCullagh, 1983) and in generalized estimating equations for models for clustered and longitudinal data (Liang and Zeger, 1986). In all of the above cases, the construction of confidence intervals can be done using the profiles of appropriate objective functions in the same way as the likelihood profiles. For example, in the case of bias-reduction in logistic regression via maximum penalized likelihood, Heinze and Schemper (2002) suggest to use the profiles of the penalized likelihood, and when estimation is via a set of estimating equations Lindsay and Qu (2003) suggest the use of profiles of appropriate quadratic score functions.

profileModel is an **R** package that provides tools to calculate, evaluate, plot and use for inference the profiles of *arbitrary* objectives for arbitrary fitted models with *linear predictors*. It allows the developers of fitting procedures to use these capabilities by simply writing a small function for the appropriate objective to be profiled. Furthermore, it provides an alternative to already implemented profiling methods, such as `profile.glm` and `profile.polr`, by extending and improving their capabilities (see, the incidence of leaf-blotch on barley example in the Examples section).

In its current version (0.5-3), the **profileModel** package has been tested and is known to work for fitted objects resulting from `lm`, `glm`, `polr`, `gee`, `geeglm`, `brglm`, `brlr`, `BTm` and `survreg`.

Supported fitted objects

profileModel aims at generality of application and thus it is designed to support most of the classes of fitted objects that are constructed according to the specifications of a fitted object as given in Chambers and Hastie (1991, Chapter 2). Specifically, the supported classes of fitted objects (`object`) should comply with the following:

1. The parameters of the model to be fitted have to be related through a linear combination of parameters and covariates. That is, only models with *linear predictors* are supported.

2. The fitting procedure that resulted in `object` has to support `offset` in `formula`. Also, `object$call` has to be the call that generated `object`.

3. `object` has to be an object which supports the method `coef` and which has `object$terms` with the same meaning as, for example, in `lm` and `glm`. `coef(object)` has to be a *vector* of coefficients where each component corresponds to a column of the model matrix

   ```
   > mf <- model.frame(object$terms,
   +     data = eval(object$call$data))
   > model.matrix(object$terms, mf,
   +     contrasts = object$contrasts)
   ```

   or maybe just `model.matrix(object)`, for fitted objects that support the `model.matrix` method.

   Exceptions to this are objects returned by the `BTm` function of the **BradleyTerry** package, where some special handling of the required objects takes place.

   Note that any or both of `data` and `contrasts` could be `NULL`. This depends on whether the `data` argument has been supplied to the fitting function and whether `object$contrasts` exists.

 ISSN 1609-3631

4. Support for a `summary` method is optional. The `summary` method is only used for obtaining the estimated asymptotic standard errors associated to the coefficients in `object`. If these standard errors are not in `coef(summary(object))[,2]`, they can be supplied by the user.

The profileModel objective functions

Given a fitted object of p parameters, the $p-1$ dimensional restricted fit is an object resulting from the following procedure:

1. Restrict a parameter β_r, $r \in \{1,\ldots,p\}$, to a specific scalar c.
2. Calculate cx_r, where x_r is the model matrix column that corresponds to β_r.
3. Estimate the remaining $p-1$ parameters with cx_r as an additional offset in the model formula.

The profiles of an objective should depend on the restricted fit and a **profileModel** objective function should be defined as such.

For example, if `object` was the result of a `glm` call and `restrFit` is the object corresponding to the restricted fit for $\beta_r = c$ for some $r \in \{1,\ldots,p\}$ then an appropriate objective function for the profile likelihood is

```
profDev <- function(restrFit, dispersion)
   restrFit$deviance/dispersion
```

where the `dispersion` argument has to be defined in the relevant **R** functions calling `profDev`. Once `profDev` is passed to the functions in `profileModel` i) the restricted fits for an appropriate grid of c-values are obtained using `glm` with the appropriate offsets and covariates in its `formula` argument and ii) the differences

```
profDev(restrFit) - profDev(object)
```

are calculated for each restricted fit. Note that, in the current case, where the fitting procedure is maximum likelihood and the objective is the deviance, the above difference is the log-likelihood ratio that is usually used for testing the hypothesis $\beta_r = c$, i.e.,

$$2\left\{l(\hat{\beta}_1,\ldots,\hat{\beta}_p) - l(\hat{\beta}^c_1,\ldots,\hat{\beta}^c_{r-1},c,\hat{\beta}^c_{r+1},\ldots,\hat{\beta}^c_p)\right\}, \quad (1)$$

where $\hat{\beta}^c_t$ $(t = 1,\ldots,r-1,r+1,\ldots,p)$ is the restricted maximum likelihood estimate when $\beta_r = c$, and $l(.)$ is the log-likelihood function.

As an illustrative example consider

```
> m1 <- example(birthwt, "MASS")$value
> library(profileModel)
> prof1.m1 <- profileModel(fitted = m1,
+     objective = profDev,
+     dispersion = 1)
```

Here, we use the default behaviour of the `profileModel` function; `prof1.m1$profiles` is a list of matrices, one for each estimated parameter in `m1`. The first column of each matrix contains 10 values of the corresponding parameter with the smallest and largest values being 5 estimated standard errors on the left and on the right, respectively, of the maximum likelihood estimate of the parameter in `m1`. The second column contains the values of (1) that correspond to the grid of values in the first column.

The profileModel function and class

The primary function of the **profileModel** package is the profiling function with the same name, which provides the following two main features.

Profiling the objective over a specified grid of values for the parameters of interest.

This feature can be used with generic objectives irrespective of their shape. It is most useful for the study of the shape of the profiles in a given range of parameter values. As an illustration consider the following:

```
> prof2.m1 <- update(prof1.m1,
+     which = c("age", "raceblack", "ftv1"),
+     grid.bounds = c(-2, 0, 0, 2, -1, 1),
+     gridsize = 50)
> plot(prof2.m1)
```

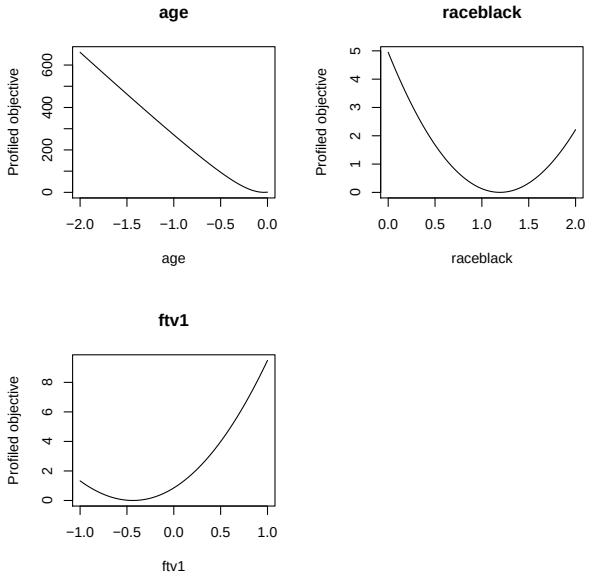


Figure 1: Profiling the log-likelihood ratio statistic over a grid of values.

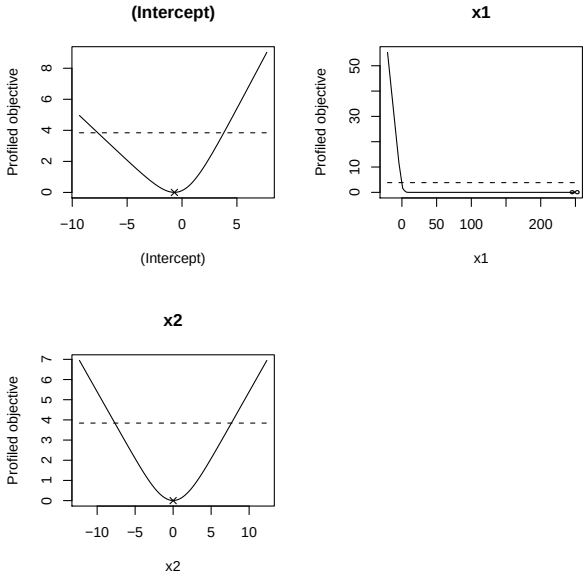


Figure 2: Profiling until the profile exceeds `qchisq(0.95,1)` (dashed line). The ML estimate for `x1` is infinity.

Profiling the objective until the value of the profile exceeds a given quantile value for the parameters of interest.

This feature relates to convex objectives. Restricted fits are calculated on the left and on the right of the estimate until the value of the profile exceeds a given quantile value for the parameters of interest. In contrast with the default `profile` methods, in **profileModel** the size of the steps to be taken on the left and on the right of the estimate depends on the shape of the profile curve through estimates of the slope of the tangents to the curve. Specifically, the steps are smaller when the profile curve seems to change fast. In this way any asymptotes in either direction can be detected (see the documentation of the `profileModel` function for detailed information on the calculation of the steps). For example, consider

```
> y <- c(1/3, 1/3, 1, 1)
> x1 <- c(0, 0, 1, 1) ;  x2 <- c(0, 1, 0, 1)
> m2 <- glm(y ~ x1 + x2, binomial)
> prof.m2 <- profileModel(m2,
+     objective = profDev,
+     quantile = qchisq(0.95, 1),
+     gridsize = 50, dispersion = 1)
> plot(prof.m2)
```

The `profileModel` procedure results in objects of class `profileModel`. S3 methods specific to `profileModel` objects are

- **Generics:**
  `print`, `update`
- **Plotting facilities:**
  `plot`, `pairs`
- **Construction of confidence intervals:**
  `profConfint`, `profZoom`, `profSmooth`

Construction of confidence intervals

The confidence-interval methods in the **profileModel** package relate to convex objectives. Asymptotic confidence intervals based on the profiles of the objective can be constructed using the `profConfint` method, which is a wrapper of two different methods for the computation of confidence intervals.

Spline smoothing via the `profSmooth` method

A smoothing spline is fitted on the points of the profiles in a `profileModel` object and then the endpoints of the confidence intervals are computed by finding the value of the parameter where the fitted spline has the value of the appropriate quantile. For example,

```
> profConfint(prof.m2, method="smooth")
                 Lower    Upper
(Intercept) -7.670327 3.802680
```

```
x1          -0.507256      Inf
x2          -7.668919 7.668919
```

Since `quantile = qchisq(0.95,1)` was used for `prof.m2`, these are 95% asymptotic profile confidence intervals.

The advantage of spline smoothing is that it requires minimal computation. However, by construction the method depends on the size of the profiling grid, as well as on the shape of the profiles. The endpoints of the resultant confidence intervals are most accurate when the profiles are quadratic in shape.

Binary search via the `profZoom` method

A binary search is performed that 'zooms in' on the appropriate regions until the absolute difference of the quantile from the profile value at each end point is smaller than a specified tolerance. Construction of confidence intervals in this way requires the calculation of the value of the profiles at the candidates returned by the binary search. Thus, in contrast to smoothing, this method, whereas accurate, is computationally intensive. On the same example as for spline smoothing we have

```
> profConfint(prof.m2, method="zoom",
+     endpoint.tolerance = 1e-8)
                Lower    Upper
(Intercept) -7.6703317 3.802770
x1          -0.8518235      Inf
x2          -7.6689352 7.668935
```

The smoothing method was accurate up to at least 3 decimal places for the endpoints of the confidence intervals for `(Intercept)` and `x1`. However, it failed to accurately estimate the left endpoint of the confidence interval for `x1`; the profile for `x1` in Figure 2 has an asymptote on the right.

Examples

Profile likelihood for `survreg` fitted objects

To illustrate the generality of application of the **profileModel** package we consider the example in the help file of the `survreg` function in the **survival** package.

```
> library(survival)
> m3 <- survreg(
+     Surv(futime, fustat) ~ ecog.ps + rx,
+     ovarian, dist= 'weibull', scale = 1)
```

For the parameters `(Intercept)`, `ecog.ps` and `rx` we calculate the values of the log-likelihood ratio statistic in (1), plot the profiles and obtain asymptotic confidence intervals based on the profiles. According to the rules for the construction of objectives in previous sections and to the available quantities in a `survreg` object, an appropriate **profileModel** objective could be defined as

```
> profLogLik <- function(restrFit) {
+     -2*restrFit$loglik[2]
+ }
```

In the case of a `survreg` object, the estimated asymptotic standard errors for the estimates are given by `summary(m3)$table[,2]`. We have

```
> prof.m3 <- profileModel(m3,
+     quantile=qchisq(0.95,1),
+     objective = profLogLik,
+     stdErrors = summary(m3)$table[,2],
+     gridsize=20)
```

Using binary search, the profile likelihood 95% asymptotic confidence intervals are

```
> profConfint(prof.m3,
+     method="zoom",
+     endpoint.tolerance = 1e-8)
                Lower     Upper
(Intercept)  4.5039809 9.7478481
ecog.ps     -1.6530569 0.7115453
rx          -0.5631772 1.8014250
```

Note that the 95% Wald asymptotic confidence intervals

```
> confint(m3)
                2.5 %    97.5 %
(Intercept)  4.3710056 9.5526696
ecog.ps     -1.5836210 0.7173517
rx          -0.5689836 1.7319891
```

are similar because of the almost quadratic shape of the profiles, or equivalently of the apparent linearity of the signed square roots of the profiles as can be seen in Figure 3.

```
> plot(prof.m3, signed=TRUE)
```

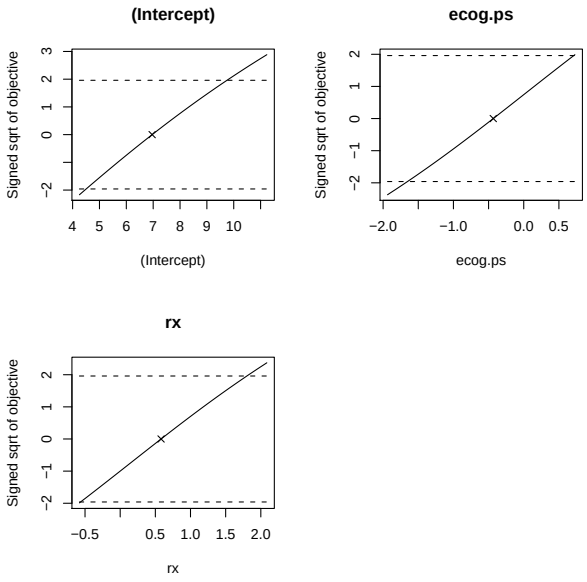


Figure 3: Signed square roots of the profiles for `m3`.

Incidence of leaf-blotch on barley

As as demonstration of the improvement over the default `profile` methods, as well as, of the generality of the **profileModel** package in terms of profiled objectives, we consider the leaf-blotch on barley study found in McCullagh and Nelder (1989, Section 9.4.2).

A linear logistic regression model with main effects is chosen to describe the variety and site effects and estimation is performed using quasi-likelihood with variance function $\mu^2(1-\mu)^2$ (see Wedderburn, 1974). The family object to be used along with `glm` is implemented through the `wedderburn` function of the **gnm** package.

```
> library(gnm)
> data(barley)
> m4 <- glm(y ~ site + variety,
+     family = wedderburn, data = barley)
```

In this case, an appropriate **profileModel** objective is the quasi-deviance (i.e. calculate the values of (1) with $l(.)$ replaced by the corresponding quasi-likelihood function).

```
> profPars <- paste("variety", c(2:9,"X"),
+     sep = "")
> prof1.m4 <- profileModel(m4,
+     objective = profDev,
+     quantile = qchisq(0.95, 1),
+     which = profPars,
+     gridsize = 20,
+     dispersion = summary(m4)$dispersion)
> cis1.m4 <- profConfint(prof1.m4)
> cis1.m4
              Lower    Upper
variety2 -1.4249881 0.5047550
variety3 -0.8653948 1.0327784
variety4  0.0086728 1.8875857
variety5  0.4234367 2.2604532
variety6  0.3154417 2.3844366
variety7  1.4351908 3.2068870
variety8  2.3436347 4.1319455
variety9  2.1698223 4.0626864
varietyX  2.9534824 4.7614612
> plot(prof1.m4, cis= cis1.m4)
```

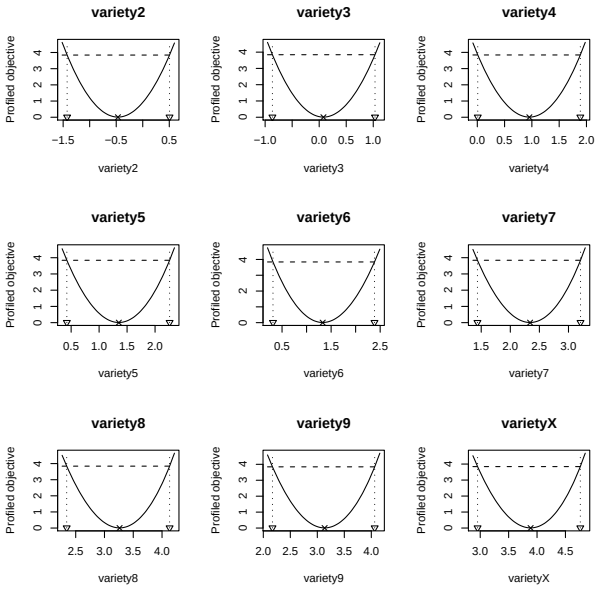


Figure 4: Profiles for `m4`.

In contrast to `profileModel`, the default `profile` method of the **MASS** package for `glm` fitted objects fails to calculate the correct profiles in this example; the profile confidence intervals using the default `profile` method are

```
> confint(m4)[profPars, ]
Waiting for profiling to be done...
               2.5 %     97.5 %
variety2 -1.42315422  0.5048776
variety3 -0.86535806  1.0328576
variety4 -0.03998811  0.8842478
variety5 -2.23635211 -2.2011092
variety6  0.31534770  2.3847282
variety7  1.41342757  1.7276810
variety8  2.31977238  2.6500143
variety9  2.15012980  2.5229588
varietyX  2.93050444  3.2755633
There were 36 warnings (use warnings()
to see them)
```

which clearly disagree with the confidence intervals obtained above (the reader is also referred to the example in the documentation of the `profConfint` function). The produced warnings give a clue for the misbehaviour; some of the restricted fits do not converge to the maximum quasi-likelihood estimates and so wrong values for the profiled objective are returned. This is most probably caused by bad starting values, which, in turn, depend on the size of the steps taken on the left and on the right of each estimate in the default `profile` method. In `profileModel`, the dependence of the size of the steps on the quasi-likelihood profiles avoids such issues.

In addition to the `plot` method, the **profileModel** package provides a `pairs` method for the construction of pairwise profile trace plots (see, Venables and Ripley, 2002, for a detailed description). This method

is a minor modification of the original `pairs` method for `profile` objects in the **MASS** package. The modification was made under the GPL licence version 2 or greater and with the permission of the original authors, in order to comply with objects of class `profileModel`.

```
> pairs(prof1.m4)
```

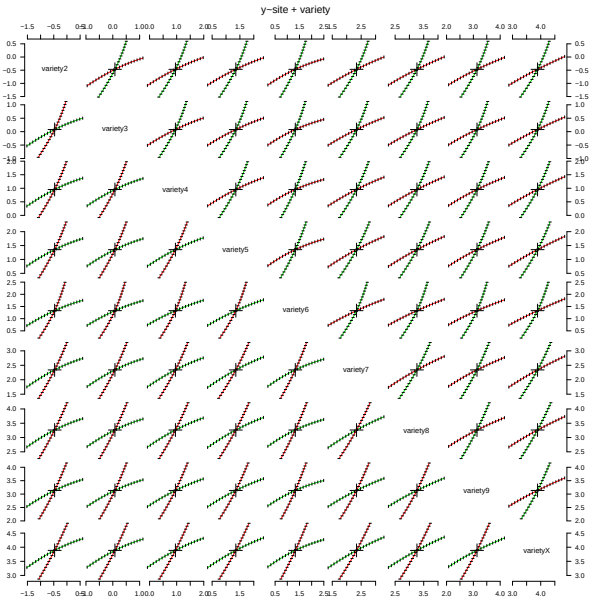


Figure 5: Pairwise profile trace plots for `m4`.

The direction of the pairwise profile trace plots in Figure 5 agrees with the fact that all the correlations amongst the variety effects are exactly $1/2$.

An alternative to the quasi-likelihood function for the construction of asymptotic confidence intervals in quasi-likelihood estimation is the quadratic form

$$Q^2(\beta) = s(\beta)^T i^{-1}(\beta) s(\beta)\,, \tag{2}$$

where $\beta = (\beta_1, \ldots, \beta_p)$ is the parameter vector, $s(\beta)$ is the vector of the quasi-score functions and $i(\beta) = \mathrm{cov}(s(\beta))$. A review on test statistics of the above form and their properties can be found in Lindsay and Qu (2003). The quadratic form (2) could be used when there is not a unique quasi-likelihood function corresponding to the set of quasi-score functions. The profile test statistic for the hypothesis $\beta_r = c$ for some $r \in \{1, \ldots, p\}$ has the form

$$Q^2(\hat\beta^c_1, \ldots, \hat\beta^c_{r-1}, c, \hat\beta^c_{r+1}, \ldots, \hat\beta^c_p) - Q^2(\hat\beta_1, \ldots, \hat\beta_p)\,, \tag{3}$$

where $\hat\beta^c_t$ $(t = 1, \ldots, r-1, r+1, \ldots, p)$ are the restricted quasi-likelihood estimates when $\beta_r = c$.

The appropriate **profileModel** objective to calculate (3) is implemented through the `RaoScoreStatistic` function, on which more details can be found in the documentation of **profileModel**. For `m4` we have

```
> prof2.m4 <- profileModel(m4,
+     objective = RaoScoreStatistic,
+     quantile = qchisq(0.95, 1),
+     which = profPars,
+     gridsize = 20, X = model.matrix(m4),
+     dispersion = summary(m4)$dispersion)
> profConfint(prof2.m4)
               Lower     Upper
variety2 -1.4781768 0.5971332
variety3 -0.9073192 1.0991394
variety4 -0.0401468 1.9064558
variety5  0.3890679 2.2472880
variety6  0.2144497 2.6012640
variety7  1.4212454 3.1476794
variety8  2.3107389 4.0685900
variety9  2.0921331 4.0503524
varietyX  2.9017855 4.6967597
```

Summary

In this article we introduce and demonstrate the main capabilities of the **profileModel** package. The described facilities, along with other complementary functions included in the package were developed with the following aims:

- **Generality:** the package provides a unified approach to profiling objectives. It supports most of the classes of fitted objects with linear predictors that are constructed according to the specifications of a fitted object as given by Chambers and Hastie (1991, Chapter 2).
- **Embedding:** the developers of current and new fitting procedures can have direct access to profiling capabilities. The only requirement is authoring a simple function that calculates the value of the appropriate objective to be profiled. Furthermore, each function of the **profileModel** package is designed to perform a very specific task. For example, `prelim.profiling` relates only to convex objectives and results in the grid of parameter values which should be used in order for the profile to cover a specific value (usually a quantile). Such a modular design enables developers to use only the relevant functions for their specific application (see, for example, the documentation of the `profile.brglm` and `confint.brglm` methods of the **brglm** package).
- **Computational stability:** all the facilities have been developed with computational stability in mind, in order to provide an alternative that improves and extends the capabilities of already available `profile` methods.

The result, we hope, is a flexible and extensible package that will be useful both to other package authors and to end-users of **R**'s statistical modelling functions.

Bibliography

J. M. Chambers and T. Hastie. *Statistical Models in S*. Chapman & Hall, 1991.

D. Firth. Bias reduction of maximum likelihood estimates. *Biometrika*, 80(1):27–38, 1993.

G. Heinze and M. Schemper. A solution to the problem of separation in logistic regression. *Statistics in Medicine*, 21:2409–2419, 2002.

K.-Y. Liang and S. L. Zeger. Longitudinal data analysis using generalized linear models. *Biometrika*, 73: 13–22, 1986.

B. G. Lindsay. Composite likelihood methods. In N. U. Prabhu, editor, *Statistical Inference from Stochastic Processes*, pages 221–239. American Mathematical Society, 1988.

B. G. Lindsay and A. Qu. Inference functions and quadratic score tests. *Statistical Science*, 18(3):394–410, 2003.

P. McCullagh. Quasi-likelihood functions. *The Annals of Statistics*, 11:59–67, 1983.

P. McCullagh and J. Nelder. *Generalized Linear Models*. Chapman and Hall, London, 2nd edition, 1989.

W. N. Venables and B. D. Ripley. Statistics complements to Modern Applied Statistics with S, 2002. URL `http://www.stats.ox.ac.uk/pub/MASS4/VR4stat.pdf`.

R. W. M. Wedderburn. Quasi-likelihood functions, generalized linear models, and the Gauss-Newton method. *Biometrika*, 61:439–447, 1974.

Ioannis Kosmidis
Department of Statistics University of Warwick
Coventry
CV4 7AL
United Kingdom
`i.kosmidis@warwick.ac.uk`

An Introduction to Text Mining in R

Ingo Feinerer

Introduction

Text mining has gained big interest both in academic research as in business intelligence applications within the last decade. There is an enormous amount of textual data available in machine readable format which can be easily accessed via the Internet or databases. This ranges from scientific articles, abstracts and books to memos, letters, online forums, mailing lists, blogs, and other communication media delivering sensible information.

Text mining is a highly interdisciplinary research field utilizing techniques from computer science, linguistics, and statistics. For the latter R is one of the leading computing environments offering a broad range of statistical methods. However, until recently, R has lacked an explicit framework for text mining purposes. This has changed with the **tm** (Feinerer, 2008; Feinerer et al., 2008) package which provides a text mining infrastructure for R. This allows R users to work efficiently with texts and corresponding meta data and transform the texts into structured representations where existing R methods can be applied, e.g., for clustering or classification.

In this article we will give a short overview of the **tm** package. Then we will present two exemplary text mining applications, one in the field of stylometry and authorship attribution, the second is an analysis of a mailing list. Our aim is to show that **tm** provides the necessary abstraction over the actual text management so that the reader can use his own texts for a wide variety of text mining techniques.

The tm package

The **tm** package offers functionality for managing text documents, abstracts the process of document manipulation and eases the usage of heterogeneous text formats in R. The package has integrated database backend support to minimize memory demands. An advanced meta data management is implemented for collections of text documents to alleviate the usage of large and with meta data enriched document sets. With the package ships native support for handling the Reuters 21578 data set, Gmane RSS feeds, e-mails, and several classic file formats (e.g. plain text, CSV text, or PDFs). The data structures and algorithms can be extended to fit custom demands, since the package is designed in a modular way to enable easy integration of new file formats, readers, transformations and filter operations. **tm** provides easy access to preprocessing and manipulation mechanisms such as whitespace removal, stemming, or conversion between file formats. Further a generic filter architecture is available in order to filter documents for certain criteria, or perform full text search. The package supports the export from document collections to term-document matrices, and string kernels can be easily constructed from text documents.

Wizard of Oz stylometry

The Wizard of Oz book series has been among the most popular children's novels in the last century. The first book was created and written by Lyman Frank Baum, published in 1900. A series of Oz books followed until Baum died in 1919. After his death Ruth Plumly Thompson continued the story of Oz books, but there was a longstanding doubt which was the first Oz book written by Thompson. Especially the authorship of the 15th book of Oz—*The Royal Book of Oz*—has been longly disputed amongst literature experts. Today it is commonly attributed to Thompson as her first Oz book, supported by several statistical stylometric analyses within the last decade.

Based on some techniques shown in the work of Binongo (2003) we will investigate a subset of the Oz book series for authorship attribution. Our data set consists of five books of Oz, three attributed to Lyman Frank Baum, and two attributed to Ruth Plumly Thompson:

The Wonderful Wizard of Oz is the first Oz book written by Baum and was published in 1900.

The Marvelous Land of Oz is the second Oz book by Baum published in 1904.

Ozma of Oz was published in 1907. It is authored by Baum and forms the third book in the Oz series.

The Royal Book of Oz is nowadays attributed to Thompson, but its authorship has been disputed for decades. It was published in 1921 and is considered as the 15th book in the series of Oz novels.

Ozoplaning with the Wizard of Oz was written by Thompson, is the 33rd book of Oz, and was published in 1939.

Most books of Oz, including the five books we use as corpus for our analysis, can be freely downloaded at the Gutenberg Project website (`http://www.gutenberg.org/`) or at the Wonderful Wizard of Oz website (`http://thewizardofoz.info/`).

The main data structure in **tm** is a *corpus* consisting of *text document*s and additional meta data. So-called *reader* functions can be used to read in texts

 ISSN 1609-3631

from various *source*s, like text files on a hard disk. E.g., following code creates the corpus `oz` by reading in the text files from the directory 'OzBooks/', utilizing a standard reader for plain texts.

```
oz <- Corpus(DirSource("OzBooks/"))
```

The directory 'OzBooks/' contains five text files in plain text format holding the five above described Oz books. So the resulting corpus has five elements representing them:

```
> oz
A text document collection with 5 text documents
```

Since our input are plain texts without meta data annotations we might want to add the meta information manually. Technically we use the `meta()` function to operate on the meta data structures of the individual text documents. In our case the meta data is stored locally in explicit slots of the text documents (`type = "local"`):

```
meta(oz, tag = "Author", type = "local") <-
   c(rep("Lyman Frank Baum", 3),
     rep("Ruth Plumly Thompson", 2))
meta(oz, "Heading", "local") <-
   c("The Wonderful Wizard of Oz",
     "The Marvelous Land of Oz",
     "Ozma of Oz",
     "The Royal Book of Oz",
     "Ozoplaning with the Wizard of Oz")
```

E.g. for our first book in our corpus this yields

```
> meta(oz[[1]])
Available meta data pairs are:
  Author       : Lyman Frank Baum
  Cached       : TRUE
  DateTimeStamp: 2008-04-21 16:23:52
  ID           : 1
  Heading      : The Wonderful Wizard of Oz
  Language     : en_US
  URI          : file
    OzBooks//01_TheWonderfulWizardOfOz.txt UTF-8
```

The first step in our analysis will be the extraction of the most frequent terms, similarly to a procedure by Binongo (2003). At first we create a so-called *term-document matrix*, that is a representation of the corpus in form of a matrix with documents as rows and terms as columns. The matrix elements are frequencies counting how often a term occurs in a document.

```
ozMatBaum <- TermDocMatrix(oz[1:3])
ozMatRoyal <- TermDocMatrix(oz[4])
ozMatThompson <- TermDocMatrix(oz[5])
```

After creating three term-document matrices for the three cases we want to distinguish—first, books by Baum, second, the Royal Book of Oz, and third, a book by Thompson—we can use the function `findFreqTerms(mat, low, high)` to compute the terms in the matrix `mat` occurring at least `low` times and at most `high` times (defaults to `Inf`).

```
baum <- findFreqTerms(ozMatBaum, 70)
royal <- findFreqTerms(ozMatRoyal, 50)
thomp <- findFreqTerms(ozMatThompson, 40)
```

The lower bounds have been chosen in a way that we obtain about 100 terms from each term-document matrix. A simple test of intersection within the 100 most common words shows that the Royal Book of Oz has more matches with the Thompson book than with the Baum books, being a first indication for Thompson's authorship.

```
> length(intersect(thomp, royal))
[1] 73
> length(intersect(baum, royal))
[1] 65
```

The next step applies a Principal Component Analysis (PCA), where we use the **kernlab** (Karatzoglou et al., 2004) package for computation and visualization. Instead of using the whole books we create equally sized chunks to consider stylometric fluctuations within single books. In detail, the function `makeChunks(corpus, chunksize)` takes a corpus and the chunk size as input and returns a new corpus containing the chunks. Then we can compute a term-document matrix for a corpus holding chunks of 500 lines length. Note that we use a *binary weighting* of the matrix elements, i.e., multiple term occurrences are only counted once.

```
ozMat <- TermDocMatrix(makeChunks(oz, 500),
           list(weighting = weightBin))
```

This matrix is the input for the Kernel PCA, where we use a standard Gaussian kernel, and request two feature dimensions for better visualization.

```
k <- kpca(as.matrix(ozMat), features = 2)
plot(rotated(k),
     col = c(rep("black", 10), rep("red", 14),
             rep("blue", 10),
             rep("yellow", 6), rep("green", 4)),
     pty = "s",
     xlab = "1st Principal Component",
     ylab = "2nd Principal Component")
```

Figure 1 shows the results. Circles in black are chunks from the first book of Oz by Baum, red circles denote chunks from the second book by Baum, and blue circles correspond to chunks of the third book by Baum. Yellow circles depict chunks from the long disputed 15th book (by Thompson), whereas green circles correspond to the 33rd book of Oz by Thompson. The results show that there is a visual correspondence between the 15th and the 33rd book (yellow and green), this means that both books tend to be authored by Thompson. Anyway, the results also unveil that

there are parts matching with books from Baum.

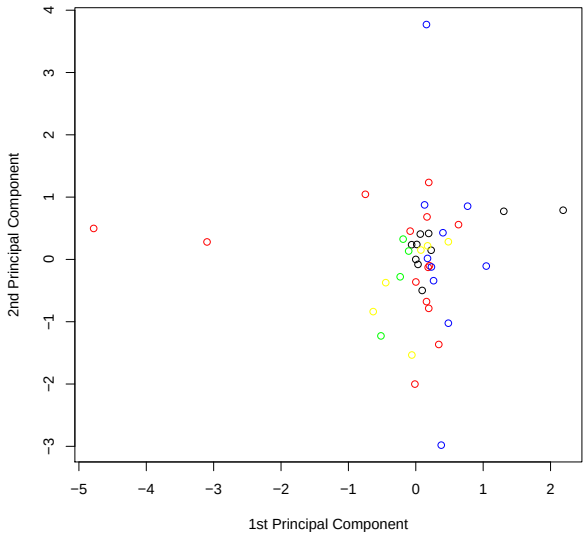


Figure 1: Principal component plot for five Oz books using 500 line chunks.

Similarly, we can redo this plot using chunks of 100 lines. This time we use a weighting scheme called *term frequency inverse document frequency*, a weighting very common in text mining and information retrieval.

```
ozMat <- TermDocMatrix(makeChunks(oz, 100),
          list(weighting = weightTfIdf))
```

Figure 2 shows the corresponding plot using the same colors as before. Again we see that there is a correspondence between the 15th book and the 33rd book by Thompson (yellow and green), but also matching parts with books by Baum.

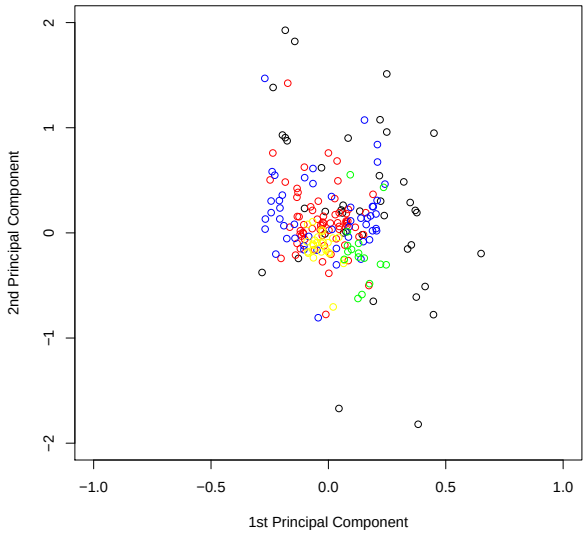


Figure 2: Principal component plot for five Oz books using 100 line chunks.

R-help mailing list analysis

Our second example deals with aspects of analyzing a mailing list. As data set we use the R-help mailing list since its archives are publicly available and can be accessed via an RSS feed or can be downloaded in mailbox format. For the RSS feed we can use the Gmane (Ingebrigtsen, 2008) service, so e.g., with

```
rss <- Corpus(GmaneSource("http://rss.gmane.org/
gmane.comp.lang.r.general"))
```

we get the latest news from the mailing list. **tm** ships with a variety of sources for different purposes, including a source for RSS feeds in the format delivered by Gmane. The source automatically uses a reader function for newsgroups as it detects the RSS feed structure. This means that the meta data in the e-mail headers is extracted, e.g., we obtain

```
> meta(rss[[1]])
Available meta data pairs are:
  Author       : Peter Dalgaard
  Cached       : TRUE
  DateTimeStamp: 2008-04-22 08:16:54
  ID           : http://permalink.gmane.org/
gmane.comp.lang.r.general/112060
  Heading      : R 2.7.0 is released
  Language     : en_US
  Origin       : Gmane Mailing List Archive
  URI          : file http://rss.gmane.org/
gmane.comp.lang.r.general UTF-8
```

for a recent mail announcing the availability of R 2.7.0.

If we want to work with a bigger collection of mails we use the mailing list archives in mailbox format. For demonstration we downloaded the file '2008-January.txt.gz' from `https://stat.ethz.ch/pipermail/r-help/`. In the next step we convert the single file containing all mails to a collection of files each containing a single mail. This is especially useful when adding new files or when lazy loading of the mail corpora is intended, i.e., when the loading of the mail content into memory is deferred until it is first accessed. For the conversion we use the function `convertMboxEml()` which extracts the R-help January 2008 mailing list postings to the directory 'Mails/2008-January/':

```
convertMboxEml(
  gzfile("Mails/2008-January.txt.gz"),
  "Mails/2008-January/")
```

Then we can create the corpus using the created directory. This time we explicitly define the reader to be used as the directory source cannot infer automatically the internal structures of its files:

```
rhelp <- Corpus(DirSource("Mails/2008-January/"),
            list(reader = readNewsgroup))
```

The newly created corpus `rhelp` representing the January archive contains almost 2500 documents:

```
> rhelp
A text document collection with 2486 documents
```

Then we can perform a set of *transformation*s, like converting the e-mails to plain text, strip extra whitespace, convert the mails to lower case, or remove e-mail signatures (i.e., text after a `--␣` (two hyphens, followed by a space) mark).

```
rhelp <- tmMap(rhelp, asPlain)
rhelp <- tmMap(rhelp, stripWhitespace)
rhelp <- tmMap(rhelp, tmTolower)
rhelp <- tmMap(rhelp, removeSignature)
```

This simple preprocessing already enables us to work with the e-mails in a comfortable way, concentrating on the main content in the mails. This way it circumvents the manual handling and parsing of the internals of newsgroup postings.

Since we have the meta data available we can find out who wrote the most postings during January 2008 in the R-help mailing list. We extract the author information from all documents in the corpus and normalize multiple entries (i.e., several lines for the same mail) to a single line:

```
authors <- lapply(rhelp, Author)
authors <- sapply(authors, paste, collapse = " ")
```

Then we can easily find out the most active writers:

```
> sort(table(authors), decreasing = TRUE)[1:3]
Gabor Grothendieck 100
 Prof Brian Ripley  97
    Duncan Murdoch  63
```

Finally we perform a full text search on the corpus. We want to find out the percentage of mails dealing with problems. In detail we search for those documents in the corpus explicitly containing the term `problem`. The function `tmIndex()` filters out those documents and returns their index where the full text search matches.

```
p <- tmIndex(rhelp, FUN = searchFullText,
             "problem", doclevel = TRUE)
```

The return variable `p` is a logical vector of the same size as the corpus indicating for each document whether the search has matched or not. So we obtain

```
> sum(p) / length(rhelp)
[1] 0.2373290
```

as the percentage of explicit problem mails in relation to the whole corpus.

Outlook

The **tm** package provides the basic infrastructure for text mining applications in R. However, there are open challenges for future research: First, larger data sets easily consist of several thousand documents resulting in large term-document matrices. This causes a severe memory problem as soon as dense data structures are computed from sparse term-document matrices. Anyway, in many cases we can significantly reduce the size of a term-document matrix by either removing stopwords, i.e., words with low information entropy, or by using a controlled vocabulary. Both techniques are supported by **tm** via the `stopwords` and `dictionary` arguments for the `TermDocMatrix()` constructor. Second, operations on large corpora are time consuming and should be avoided as far as possible. A first solution resulted in so-called *lazy transformation*s, which materialize operations only when documents are later accessed, but further improvement is necessary. Finally, we have to work on better integration with other packages for natural language processing, e.g., with packages for tag based annotation.

Acknowledgments

I would like to thank Kurt Hornik, David Meyer, and Vince Carey for their constructive comments and feedback.

Bibliography

J. N. G. Binongo. Who wrote the 15th book of Oz? An application of multivariate analysis to authorship attribution. *Chance*, 16(2):9–17, 2003.

I. Feinerer. *tm: Text Mining Package*, 2008. URL `http://CRAN.R-project.org/package=tm`. R package version 0.3-1.

I. Feinerer, K. Hornik, and D. Meyer. Text mining infrastructure in R. *Journal of Statistical Software*, 25(5):1–54, March 2008. ISSN 1548-7660. URL `http://www.jstatsoft.org/v25/i05`.

L. M. Ingebrigtsen. Gmane: A mailing list archive, 2008. URL `http://gmane.org/`.

A. Karatzoglou, A. Smola, K. Hornik, and A. Zeileis. kernlab – an S4 package for kernel methods in R. *Journal of Statistical Software*, 11(9):1–20, 2004. URL `http://www.jstatsoft.org/v11/i09/`.

Ingo Feinerer
Wirtschaftsuniversität Wien, Austria
`feinerer@logic.at`

animation: A Package for Statistical Animations

by Yihui Xie and Xiaoyue Cheng

The mastery of statistical ideas has always been a challenge for many who study statistics. The power of modern statistical software, taking advantage of advances in theory and in computer systems, both adds to those challenges and offers new opportunities for creating tools that will assist learners. The **animation** package (Xie, 2008) uses graphical and other animations to communicate the results of statistical simulations, giving meaning to abstract statistical theory. From our own experience, and that of our classmates, we are convinced that such experimentation can assist the learning of statistics. Often, practically minded data analysts try to bypass much of the theory, moving quickly into practical data analysis. For such individuals, use of animation to help understand simulations may make their use of statistical methods less of a "black box".

Introduction

Animation is the rapid display of a sequence of 2D or 3D images to create an illusion of movement. Persistence of vision creates the illusion of motion. Figure 1 shows a succession of frames that might be used for a simple animation – here an animation that does not serve any obvious statistical purpose.

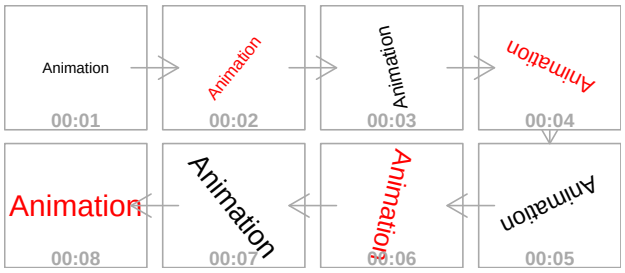


Figure 1: This sequence of frames, displayed in quick succession, creates an illusion of movement.

An animation consists of multiple image frames, which can be designed to correspond to the successive steps of an algorithm or of a data analysis. Thus far, we have sought to:

- build a gallery of statistical animations, including some that have a large element of novelty;
- provide convenient functions to generate animations in a variety of formats including HTML/JS (JavaScript), GIF/MPEG and SWF (Flash);
- make the package extensible, allowing users to create their own animations.

Design and features

It is convenient to create animations using the R environment. The code below illustrates how the functions in the **animation** package produce animations. In this case, the output is similar to the animation on the web page obtained by going to `http://animation.yihui.name/animation:start#method`, as indicated in the first line of the following code:

```
# animation:start#method
clr = rainbow(360)
for (i in 1:360) {
    plot(1, ann = F, type = "n", axes = F)
    text(1, 1, "Animation", srt = i, col =
        clr[i], cex = 7 * i/360)
    Sys.sleep(0.01)
}
```

High-level and low-level plotting commands, used along with various graphical devices described in Murrell (2005), make it straightforward to create and display animation frames. Within an R session, any available graphics device (for 'Windows', 'Linux', 'MacOS X', etc) can be used for display. Any R graphics system can be used (**graphics**, **grid**, **lattice**, **ggplot**, etc).

Image formats in which animations can be saved (and shared) include, depending on the computer platform, `png()`, `jpeg()` and `pdf()`. For display separately from an R session, possible formats include (1) HTML and JS, (2) GIF/MPEG, or (3) Flash. All that is needed is (1) a browser with JavaScript enabled, (2) an image viewer (for GIF) / a media player (for MPEG), or (3) a Flash viewer for the relevant animation format.

The basic schema for all animation functions in the package is:

```
ani.fun <- function(args.for.stat.method,
    args.for.graphics, ...) {
    {stat.calculation.for.preparation.here}
    i = 1
    while (i <= ani.options("nmax") &
        other.conditions.for.stat.method) {
        {stat.calculation.for.animation}
        {plot.results.in.ith.step}
        # pause for a while in this step
        Sys.sleep(ani.options("interval"))
        i = i + 1
    }
```

```
    # (i - 1) frames produced in the loop
    ani.options("nmax") = i - 1
    {return.something}
}
```

As the above code illustrates, an animation consists of successive plots with pauses inserted as appropriate. The function `ani.options()`, modeled on the function `options()`, is available to set or query animation parameters. For example, `nmax` sets the maximum number of steps in a loop, while `interval` sets the duration for the pause in each step. The following demonstrates the use of `ani.options()` in creating an animation of Brownian Motion. (As with all code shown in this paper, a comment on the first line of code has the name of the web page, in the root directory `http://animation.yihui.name`, where the animation can be found.)

```
# prob:brownian_motion
library(animation)
# 100 frames of brownian motion
ani.options(nmax = 100)
brownian.motion(pch = 21, cex = 5,
    col = "red", bg = "yellow")

# faster: interval = 0.02
ani.options(interval = 0.02)
brownian.motion(pch = 21, cex = 5,
    col = "red", bg = "yellow")
```

Additional software is not required for HTML/JS, as the pages are generated by writing pure text (HTML/JS) files. The functions `ani.start()` and `ani.stop()` do all that is needed. Creation of animations in GIF/MPEG or SWF formats requires third-party utilities such as ImageMagick[1] or SWF Tools[2]. We can use `saveMovie()` and `saveSWF()` to create GIF/MPEG and Flash animations, respectively.

The four examples that now follow demonstrate the four sorts of animations that are supported.

```
##########################################
# (1) Animations inside R windows graphics
# devices: Bootstrapping
oopt = ani.options(interval = 0.3, nmax = 50)
boot.iid()
ani.options(oopt)

##########################################
# (2) Animations in HTML pages: create an
# animation page for the Brownian Motion in
# the tempdir() and auto-browse it
oopt = ani.options(interval=0.05, nmax=100,
  title = "Demonstration of Brownian Motion",
  description = "Random walk on the 2D plane:
  for each point (x, y), x = x + rnorm(1)
  and y = y + rnorm(1).", outdir = tempdir())
ani.start()
opar = par(mar = c(3, 3, 2, 0.5),
  mgp = c(2, .5, 0), tcl = -0.3,
  cex.axis = 0.8, cex.lab = 0.8, cex.main = 1)
brownian.motion(pch = 21, cex = 5, col = "red",
    bg = "yellow")
par(opar)
ani.stop()
ani.options(oopt)

##########################################
# (3) GIF animations (require ImageMagick!)
oopt = ani.options(interval = 0, nmax = 100)
saveMovie({brownian.motion(pch = 21, cex = 5,
    col = "red", bg = "yellow")},
    interval = 0.05, outdir = getwd(),
    width = 600, height = 600)
ani.options(oopt)

##########################################
# (4) Flash animations (require SWF Tools!)
oopt = ani.options(nmax = 100, interval = 0)
saveSWF(buffon.needle(type = "S"),
    para = list(mar = c(3, 2.5, 1, 0.2),
    pch = 20, mgp = c(1.5, 0.5, 0)),
    dev = "pdf", swfname = "buffon.swf",
    outdir = getwd(), interval = 0.1)
ani.options(oopt)
```

Additionally, users can write their custom functions that generate animation frames, then use `ani.start()`, `ani.stop()`, `saveMovie()` and `saveSWF()` as required to create animations that can be viewed outside R.[3] The following commands define a function `jitter.ani()` that demonstrates the jitter effect when plotting discrete variables. This helps make clear the randomness of jittering.

```
# animation:example_jitter_effect
# function to show the jitter effect
jitter.ani <- function(amount = 0.2, ...) {
    x = sample(3, 300, TRUE, c(.1, .6, .3))
    y = sample(3, 300, TRUE, c(.7, .2, .1))
    i = 1
    while (i <= ani.options("nmax")) {
        plot(jitter(x, amount = amount),
            jitter(y, amount = amount),
            xlim = c(0.5, 3.5),
            ylim = c(0.5, 3.5),
            xlab = "jittered x",
            ylab = "jittered y",
            panel.first = grid(), ...)
        points(rep(1:3, 3), rep(1:3, each =
            3), cex = 3, pch = 19)
        Sys.sleep(ani.options("interval"))
        i = i + 1
```

[1] Use the command `convert` to convert multiple image frames into single GIF/MPEG files.
[2] The utilities `png2swf`, `jpeg2swf` and `pdf2swf` can convert PNG, JPEG and PDF files into Flash animations
[3] Very little change to the code is needed in order to change to a different output format.

```
    }
}

# HTML/JS
ani.options(nmax = 30, interval = 0.5,
    title = "Jitter Effect")
ani.start()
par(mar = c(5, 4, 1, 0.5))
jitter.ani()
ani.stop()

# GIF
ani.options(interval = 0)
saveMovie(jitter.ani(), interval = 0.5,
    outdir = getwd())

# Flash
saveSWF(jitter.ani(), interval = 0.5,
    dev = "png", outdir = getwd())
```

Animation examples

Currently, the **animation** package has about 20 simulation options. The website `http://animation.yihui.name` has several HTML pages of animations. These illustrate many different statistical methods and concepts, including the K-Means cluster algorithm (`mvstat:k-means_cluster_algorithm`), the Law of Large Numbers (`prob:law_of_large_numbers`), the Central Limit Theorem (`prob:central_limit_theorem`), and several simulations of famous experiments including Buffon's Needle (`prob:buffon_s_needle`).

Two examples will now be discussed in more detail – coin flips and the *k-nearest neighbor algorithm* (*k*NN).

Example 1 (Coin Flips) Most elementary courses in probability theory use coin flips as a basis for some of the initial discussion. While teachers can use coins or dice for simulations, these quickly become tedious, and it is convenient to use computers for extensive simulations. The function `flip.coin()` is designed for repeated coin flips (die tosses) and for displaying the result. Code for a simple simulation is:

```
# prob:flipping_coins
flip.coin(faces = c("Head", "Stand", "Tail"),
    type = "n", prob = c(0.45, 0.1, 0.45),
    col =c(1, 2, 4))
```

Suppose the possible results are: 'Head', 'Tail', or just 'Stand' on the table (which is rare but not impossible), and the corresponding probabilities are 0.45, 0.1, and 0.45. Figure 2 is the final result of a simulation. The frequencies (0.42, 0.14, and 0.44) are as close as might be expected to the true probabilities.

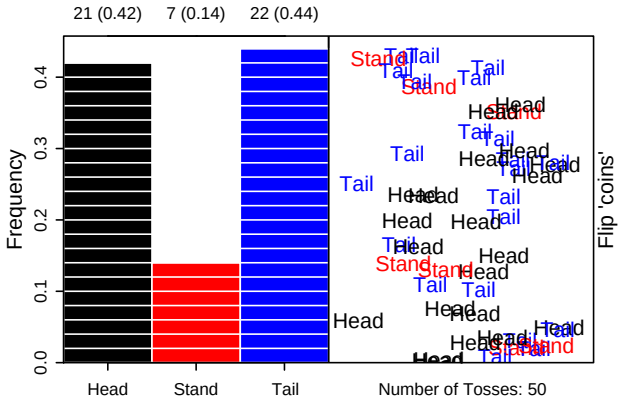


Figure 2: Result of flipping a coin 50 times. The right part of the graph shows the process of flipping, while the left part displays the result.

Example 2 (*k*-Nearest Neighbor Algorithm) The *k*-nearest neighbor algorithm is among the simplest of all machine learning algorithms. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common amongst its *k* nearest neighbors.

Basically the *k*NN algorithm has four steps:

1. locate the test point (for which we want to predict the class label);
2. compute the distances between the test point and all points in the training set;
3. find the *k* shortest distances and the corresponding training set points;
4. classify by majority vote.

The function `knn.ani()` gives an animated demonstration (i.e. the four steps above) of classification using the *k*NN algorithm. The following command uses the default arguments[4] to show the animation:

```
# dmml:k-nearest_neighbour_algorithm
knn.ani()
```

Figure 3 is a two-dimensional demonstration of the *k*NN algorithm. Points with unknown classes are marked as '?'. The algorithm is applied to each test point in turn, assigning the class ○ or △.

Our initial motivation for writing this package came in large part from the experience of students in a machine learning class. We noticed that when the lecturer described the *k*-NN algorithm, mentioning also *k*-fold cross-validation, many students confused the two '*k*'s. We therefore designed animations that were intended to help students understand the basic

[4]Note that the arguments of most animation functions have defaults.

ideas behind various machine learning algorithms. Judging from the response of the audience, our efforts proved effective.

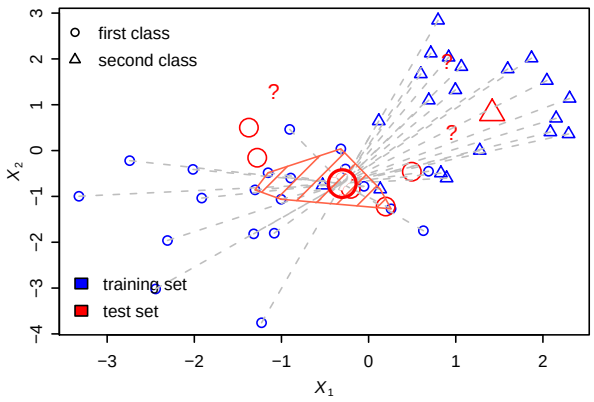


Figure 3: Demonstration of the *k*-nearest neighbor algorithm. The training and test set are denoted with different colors, and the different classes are denoted by different symbols. For each test point, there will be four animation frames. In this plot there are 8 circles and 2 triangles; hence the class '○' is assigned.

There are many other demonstrations in this package. Figure 4 is the third frame of a demonstration for *k*-fold cross-validation (`cv.ani()`). In that animation, the meaning of *k* is quite clear. No-one would confuse it with the *k* in *k*NN even when these two topics are talked about together. (Web page: `dmml:k-fold_cross-validation`)

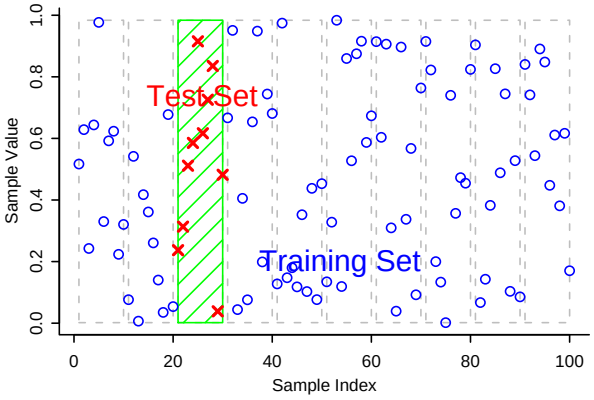


Figure 4: An animation frame from the demonstration of *k*-fold cross-validation. First, points are partitioned into 10 subsets. Each subset will be used in turn as the test set. In the animation, the subsets are taken in order from left to right.

Other examples that are on the website include the Newton-Raphson method (see Figure 5 and web page `compstat:newton_s_method`) and the Gradient Descent algorithm (see Figure 6 and web page: `compstat:gradient_descent_algorithm`).

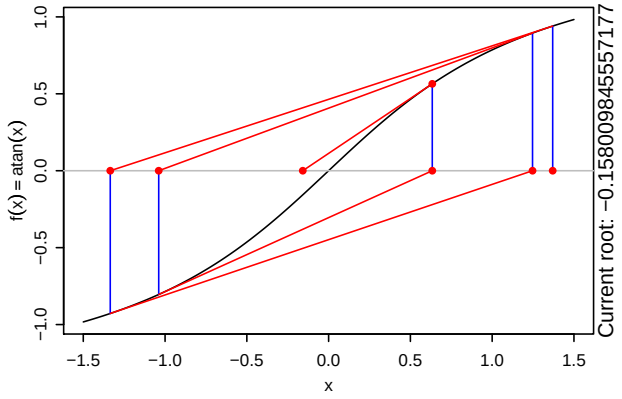


Figure 5: This is the fifth animation frame from the demonstration of the Newton-Raphson method for root-finding. The tangent lines will "guide" us to the final solution.

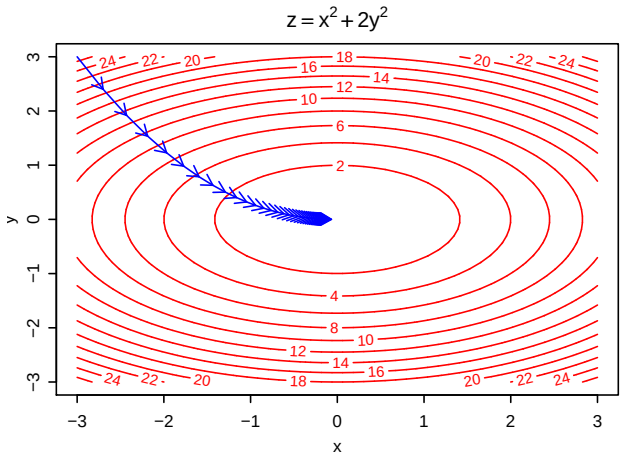


Figure 6: The final frame of a demonstration of the Gradient Descent algorithm. The arrows move from the initial solution to the final solution.

Summary

The R **animation** package is designed to convert statistical ideas into animations that will assist the understanding of statistical concepts, theories and data analysis methods. Animations can be displayed both inside R and outside R.

Further development of this package will include: (1) addition of further animations; (2) development of a graphical user interface GUI (using `Tcl/Tk` or `gWidgets`, etc), for the convenience of users.

Acknowledgment

We'd like to thank John Maindonald for his invaluable suggestions for the development of the **animation** package.

Bibliography

P. Murrell. *R Graphics*. Chapman & Hall/CRC, 2005.

Y. Xie. *animation: Demonstrate Animations in Statistics*, 2008. URL `http://animation.yihui.name`. R package version 1.0-1.

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

Xiaoyue Cheng
School of Statistics, Room 1037, Mingde Main Building, Renmin University of China, Beijing, 100872, China
`chengxiaoyue@gmail.com`

The VGAM Package

by Thomas W. Yee

Introduction

The **VGAM** package implements several large classes of regression models of which *vector generalized linear and additive models* (VGLMs/VGAMs) are most commonly used (Table 1). The primary key words are maximum likelihood estimation, iteratively reweighted least squares (IRLS), Fisher scoring and additive models. Other subsidiary concepts are reduced rank regression, constrained ordination and vector smoothing. Altogether the package represents a broad and unified framework.

Written in S4 (Chambers, 1998), the **VGAM** modelling functions are used in a similar manner as `glm()` and Trevor Hastie's `gam()` (in **gam**, which is very similar to his S-PLUS version). Given a `vglm()/vgam()` object, standard generic functions such as `coef()`, `fitted()`, `predict()`, `summary()`, `vcov()` are available. The typical usage is like

```
vglm(yvector ~ x2 + x3 + x4,
     family = VGAMfamilyFunction,
     data = mydata)
vgam(ymatrix ~ s(x2) + x3,
     family = VGAMfamilyFunction,
     data = mydata)
```

Many models have a multivariate response, and therefore the LHS of the formula is often a matrix (otherwise a vector). The function assigned to the `family` argument is known as a **VGAM** *family function*. Table 2 lists some widely used ones.

The scope of **VGAM** is very broad; it potentially covers a wide range of multivariate response types and models, including univariate and multivariate distributions, categorical data analysis, quantile and expectile regression, time series, survival analysis, extreme value analysis, mixture models, correlated binary data, bioassay data and nonlinear least-squares problems. Consequently there is overlap with many other R packages. **VGAM** is designed to be as general as possible; currently over 100 **VGAM** family functions are available and new ones are being written all the time.

Basically, VGLMs model each parameter, transformed if necessary, as a linear combination of the explanatory variables. That is,

$$g_j(\theta_j) \;=\; \eta_j \;=\; \boldsymbol{\beta}_j^T \boldsymbol{x} \tag{1}$$

where g_j is a parameter link function. This idea emerges in the next section. VGAMs extend (1) to

$$g_j(\theta_j) \;=\; \eta_j \;=\; \sum_{k=1}^{p} f_{(j)k}(x_k), \tag{2}$$

i.e., an additive model for each parameter.

Two examples

To motivate **VGAM** let's consider two specific examples: the negative binomial distribution and the proportional odds model. These are often fitted using `glm.nb()` and `polr()` respectively, both of which happen to reside in the **MASS** package. We now attempt to highlight some advantages of using **VGAM** to fit these models—that is, to describe some of the VGLM/VGAM framework.

Negative binomial regression

A negative binomial random variable Y has a probability function that can be written as

$$P(Y=y) \;=\; \binom{y+k-1}{y} \left(\frac{\mu}{\mu+k}\right)^y \left(\frac{k}{k+\mu}\right)^k$$

where $y = 0, 1, 2, \ldots$, and parameters μ $(= E(Y))$ and k are positive. The quantity $1/k$ is known as the dispersion parameter, and the Poisson distribution is the limit as k approaches infinity. The negative binomial is commonly used to handle overdispersion with respect to the Poisson distribution because $\mathrm{Var}(Y) = \mu + \mu^2/k > \mu$.

The **MASS** implementation is restricted to a intercept-only estimate of k. For example, one cannot fit $\log k = \beta_{(2)1} + \beta_{(2)2} x_2$, say. In contrast, **VGAM** can fit

$$\begin{aligned} \log \mu \;&=\; \eta_1 \;&=\; \boldsymbol{\beta}_1^T \boldsymbol{x}, \\ \log k \;&=\; \eta_2 \;&=\; \boldsymbol{\beta}_2^T \boldsymbol{x}. \end{aligned}$$

by using `negbinomial(zero=NULL)` in the call to `vglm()`. Note that it is natural for any positive parameter to have the log link as the default.

There are also variants of the negative binomial in the **VGAM** package, e.g., the zero-inflated and zero-altered versions; see Table 2.

Proportional odds model

The proportional odds model (POM) for an ordered factor Y taking levels $\{1, 2, \ldots, M+1\}$ may be written

$$\mathrm{logit}\, P(Y \le j | \boldsymbol{x}) \;=\; \eta_j(\boldsymbol{x}) \tag{3}$$

where $\boldsymbol{x} = (x_1, \ldots, x_p)^T$ with $x_1 = 1$ denoting an intercept, and

$$\eta_j(\boldsymbol{x}) \;=\; \beta_{(j)1} + \boldsymbol{\beta}_*^T \boldsymbol{x}_{(-1)}, \quad j = 1, \ldots, M. \tag{4}$$

Here, $\boldsymbol{x}_{(-1)}$ is $\boldsymbol{x}$ with the first element deleted. Standard references for the POM include McCullagh and Nelder (1989) and Agresti (2002).

η	Model	S function	Reference
$\mathbf{B}_1^T\boldsymbol{x}_1 + \mathbf{B}_2^T\boldsymbol{x}_2\ (= \mathbf{B}^T\boldsymbol{x})$	VGLM	`vglm()`	Yee and Hastie (2003)
$\mathbf{B}_1^T\boldsymbol{x}_1 + \sum_{k=p_1+1}^{p_1+p_2} \mathbf{H}_k f_k^*(x_k)$	VGAM	`vgam()`	Yee and Wild (1996)
$\mathbf{B}_1^T\boldsymbol{x}_1 + \mathbf{A}\,\boldsymbol{\nu}$	RR-VGLM	`rrvglm()`	Yee and Hastie (2003)
$\mathbf{B}_1^T\boldsymbol{x}_1 + \mathbf{A}\,\boldsymbol{\nu} + \begin{pmatrix} \boldsymbol{\nu}^T\mathbf{D}_1\boldsymbol{\nu} \\ \vdots \\ \boldsymbol{\nu}^T\mathbf{D}_M\boldsymbol{\nu} \end{pmatrix}$	QRR-VGLM	`cqo()`	Yee (2004a)
$\mathbf{B}_1^T\boldsymbol{x}_1 + \sum_{r=1}^{R} f_r(\nu_r)$	RR-VGAM	`cao()`	Yee (2006)

Table 1: A summary of **VGAM** and its framework. The latent variables $\boldsymbol{\nu} = \mathbf{C}^T\boldsymbol{x}_2$, or $\nu = \boldsymbol{c}^T\boldsymbol{x}_2$ if rank $R = 1$. Here, $\boldsymbol{x}^T = (\boldsymbol{x}_1^T, \boldsymbol{x}_2^T)$. Abbreviations: A = additive, C = constrained, L = linear, O = ordination, Q = quadratic, RR = reduced-rank, VGLM = vector generalized linear model.

The **VGAM** family function `cumulative()` may be used to fit this model. Additionally, in contrast to `polr()`, it can also fit some useful variants/extensions. Some of these are described as follows.

(i) Nonproportional odds model

The regression coefficients $\boldsymbol{\beta}_*$ in (4) are common for all j. This is known as the so-called *parallelism* or *proportional odds* assumption. This is a theoretically elegant idea because the η_j do not cross, and therefore there are no problems with negative probabilities etc. However, this assumption is a strong one and ought to be checked for each explanatory variable.

(ii) Selecting different link functions.

VGAM is purposely extensible. The `link` argument to `cumulative()` may be assigned *any* **VGAM** link function whereas `polr()` currently provides four fixed choices. Of course, the user is responsible for choosing appropriate links, e.g., the probit and complementary loglog. Users may write their own **VGAM** link function if they wish.

(iii) *Partial proportional odds model*

An intermediary between the proportional odds and nonproportional odds models is to have some explanatory variables parallel and others not. Some authors call this a *partial proportional odds model*. As an example, suppose $p = 4$, $M = 2$ and

$$\begin{aligned} \eta_1 &= \beta_{(1)1} + \beta_{(1)2}x_2 + \beta_{(1)3}x_3 + \beta_{(1)4}x_4, \\ \eta_2 &= \beta_{(2)1} + \beta_{(1)2}x_2 + \beta_{(2)3}x_3 + \beta_{(1)4}x_4. \end{aligned}$$

Here, the parallelism assumption applies to x_2 and x_4 only. This can be achieved by

```
vglm(ymatrix ~ x2 + x3 + x4,
     cumulative(parallel = TRUE ~
                x2 + x4 - 1))
```

or equivalently,

```
vglm(ymatrix ~ x2 + x3 + x4,
     cumulative(parallel =
                FALSE ~ x3))
```

There are several other extensions that can easily be handled by the constraint matrices idea described later.

(iv) Common **VGAM** family function arguments.

Many authors define the POM as

$$\text{logit}\, P(Y \geq j+1|\boldsymbol{x}) \;=\; \eta_j(\boldsymbol{x}) \qquad (5)$$

rather than (3) because $M = 1$ coincides with logistic regression. Many **VGAM** family functions share common arguments and one of them is `reverse`. Here, setting `reverse=TRUE` will fit (5). Other common arguments include `link` (usually one for each parameter), `zero`, `parallel` and initial values of parameters.

Modelling ordinal responses in **MASS** is serviced by a "one-off" function whereas **VGAM** naturally supports several related models such as the adjacent categories and continuation/stopping ratio models; see Table 2.

General framework

In this section we describe the general framework and notation. Fuller details can be found in the references of Table 1. Although this section may be skipped on first reading, an understanding of these details is necessary to realize its full potential.

Vector generalized linear models

Suppose the observed response $\boldsymbol{y}$ is a q-dimensional vector. VGLMs are defined as a model for which the conditional distribution of $\boldsymbol{Y}$ given explanatory $\boldsymbol{x}$ is of the form

$$f(\boldsymbol{y}|\boldsymbol{x};\mathbf{B}) \;=\; h(\boldsymbol{y},\eta_1,\ldots,\eta_M) \tag{6}$$

for some known function $h(\cdot)$, where $\mathbf{B} = (\boldsymbol{\beta}_1\,\boldsymbol{\beta}_2\,\cdots\,\boldsymbol{\beta}_M)$ is a $p \times M$ matrix of unknown regression coefficients, and the jth linear predictor is

$$\eta_j \;=\; \boldsymbol{\beta}_j^T\boldsymbol{x} \;=\; \sum_{k=1}^{p}\beta_{(j)k}\,x_k, \quad j=1,\ldots,M, \tag{7}$$

where $\boldsymbol{x} = (x_1,\ldots,x_p)^T$ with $x_1 = 1$ if there is an intercept. VGLMs are thus like GLMs but allow for multiple linear predictors, and they encompass models outside the limited confines of the classical exponential family.

The η_j of VGLMs may be applied directly to parameters of a distribution rather than just to means as for GLMs. A simple example is a univariate distribution with a location parameter ξ and a scale parameter $\sigma > 0$, where we may take $\eta_1 = \xi$ and $\eta_2 = \log\,\sigma$. In general, $\eta_j = g_j(\theta_j)$ for some parameter link function g_j and parameter θ_j. In **VGAM**, there are currently over a dozen links to choose from (Table 3), of which any can be assigned to any parameter, ensuring maximum flexibility.

There is no relationship between q and M in general: it depends specifically on the model or distribution to be fitted. For example, the mixture of two normal distributions has $q = 1$ and $M = 5$.

VGLMs are estimated by IRLS. Most models that can be fitted have a log-likelihood

$$\ell \;=\; \sum_{i=1}^{n} w_i\,\ell_i \tag{8}$$

and this will be assumed here. The w_i are known positive prior weights. Let $\boldsymbol{x}_i$ denote the explanatory vector for the ith observation, for $i = 1,\ldots,n$. Then one can write

$$\boldsymbol{\eta}_i \;=\; \begin{pmatrix}\eta_1(\boldsymbol{x}_i)\\ \vdots\\ \eta_M(\boldsymbol{x}_i)\end{pmatrix} \;=\; \mathbf{B}^T\boldsymbol{x}_i \;=\; \begin{pmatrix}\boldsymbol{\beta}_1^T\boldsymbol{x}_i\\ \vdots\\ \boldsymbol{\beta}_M^T\boldsymbol{x}_i\end{pmatrix}. \tag{9}$$

In IRLS, an adjusted dependent vector $\boldsymbol{z}_i = \boldsymbol{\eta}_i + \mathbf{W}_i^{-1}\boldsymbol{d}_i$ is regressed upon a large (VLM) design matrix, with $\boldsymbol{d}_i = w_i\,\partial\ell_i/\partial\boldsymbol{\eta}_i$. The working weights $\mathbf{W}_i$ here are $w_i\mathrm{Var}(\partial\ell_i/\partial\boldsymbol{\eta}_i)$ (which, under regularity conditions, is equal to $-w_i\,E[\partial^2\ell_i/(\partial\boldsymbol{\eta}_i\,\partial\boldsymbol{\eta}_i^T)]$, called the expected information matrix or EIM), giving rise to the Fisher scoring algorithm. Fisher scoring usually has good numerical stability because the $\mathbf{W}_i$ are positive-definite over a larger region of parameter space. The price to pay for this stability is typically a slower convergence rate and less accurate standard errors (Efron and Hinkley, 1978) compared to the Newton-Raphson algorithm.

Vector generalized additive models

VGAMs provide additive-model extensions to VGLMs, that is, (7) is generalized to

$$\eta_j(\boldsymbol{x}) \;=\; \beta_{(j)1} + \sum_{k=2}^{p} f_{(j)k}(x_k), \quad j=1,\ldots,M,$$

a sum of smooth functions of the individual covariates, just as with ordinary GAMs (Hastie and Tibshirani, 1990). The $\boldsymbol{f}_k = (f_{(1)k}(x_k),\ldots,f_{(M)k}(x_k))^T$ are centered for uniqueness, and are estimated *simultaneously* using *vector smoothers*. VGAMs are thus a visual data-driven method that is well suited to exploring data, and they retain the simplicity of interpretation that GAMs possess.

In practice we may wish to constrain the effect of a covariate to be the same for some of the η_j and to have no effect for others. For example, for VGAMs, we may wish to take

$$\begin{aligned}\eta_1 &= \beta_{(1)1} + f_{(1)2}(x_2) + f_{(1)3}(x_3),\\ \eta_2 &= \beta_{(2)1} + f_{(1)2}(x_2),\end{aligned}$$

so that $f_{(1)2} \equiv f_{(2)2}$ and $f_{(2)3} \equiv 0$. For VGAMs, we can represent these models using

$$\begin{aligned}\boldsymbol{\eta}(\boldsymbol{x}) &= \boldsymbol{\beta}_{(1)} + \sum_{k=2}^{p}\boldsymbol{f}_k(x_k)\\ &= \mathbf{H}_1\,\boldsymbol{\beta}^*_{(1)} + \sum_{k=2}^{p}\mathbf{H}_k\,\boldsymbol{f}^*_k(x_k)\end{aligned} \tag{10}$$

where $\mathbf{H}_1,\mathbf{H}_2,\ldots,\mathbf{H}_p$ are known full-column rank *constraint matrices*, $\boldsymbol{f}^*_k$ is a vector containing a possibly reduced set of component functions and $\boldsymbol{\beta}^*_{(1)}$ is a vector of unknown intercepts. With no constraints at all, $\mathbf{H}_1 = \mathbf{H}_2 = \cdots = \mathbf{H}_p = \mathbf{I}_M$ and $\boldsymbol{\beta}^*_{(1)} = \boldsymbol{\beta}_{(1)}$. Like the $\boldsymbol{f}_k$, the $\boldsymbol{f}^*_k$ are centered for uniqueness. For VGLMs, the $\boldsymbol{f}_k$ are linear so that

$$\mathbf{B}^T \;=\; \begin{pmatrix}\mathbf{H}_1\boldsymbol{\beta}^*_{(1)} & \mathbf{H}_2\boldsymbol{\beta}^*_{(2)} & \cdots & \mathbf{H}_p\boldsymbol{\beta}^*_{(p)}\end{pmatrix}. \tag{11}$$

Type	Name	Description
Categorical	`acat`	Adjacent categories model
	`cratio`	Continuation ratio model
	`cumulative`	Proportional and nonproportional odds model
	`multinomial`	Multinomial logit model
	`sratio`	Stopping ratio model
	`brat`	Bradley Terry model
	`bratt`	Bradley Terry model with ties
	`ABO`	ABO blood group system
Quantile regession	`lms.bcn`	Box-Cox transformation to normality
	`alsqreg`	Asymmetric least squares (expectiles on normal distribution)
	`amlbinomial`	Asymmetric maximum likelihood—for binomial
	`amlpoisson`	Asymmetric maximum likelihood—for Poisson
	`alaplace1`	Asymmetric Laplace distribution (1-parameter)
Counts	`negbinomial`	Negative binomial
	`zipoisson`	Zero-inflated Poisson
	`zinegbinomial`	Zero-inflated negative binomial
	`zibinomial`	Zero-inflated binomial
	`zapoisson`	Zero-altered Poisson (a hurdle model)
	`zanegbinomial`	Zero-altered negative binomial (a hurdle model)
	`mix2poisson`	Mixture of two Poissons
	`pospoisson`	Positive Poisson
Discrete distributions	`logarithmic`	logarithmic
	`skellam`	Skellam (Pois(λ_1) − Pois(λ_2))
	`yulesimon`	Yule-Simon
	`zetaff`	Zeta
	`zipf`	Zipf
Continuous distributions	`betaff`	Beta (2-parameter)
	`betabinomial`	Beta-binomial
	`betaprime`	Beta-prime
	`dirichlet`	Dirichlet
	`gamma2`	Gamma (2-parameter)
	`gev`	Generalized extreme value
	`gpd`	Generalized Pareto
	`mix2normal1`	Mixture of two univariate normals
	`skewnormal1`	Skew-normal
	`vonmises`	Von Mises
	`weibull`	Weibull
Bivariate distributions	`amh`	Ali-Mikhail-Haq
	`fgm`	Farlie-Gumbel-Morgenstern
	`frank`	Frank
	`morgenstern`	Morgenstern
Bivariate binary responses	`binom2.or`	Bivariate logistic odds-ratio model
	`binom2.rho`	Bivariate probit model
	`loglinb2`	Loglinear model for 2 binary responses

Table 2: Some commonly used **VGAM** family functions. They have been grouped into types. Families in the classical exponential family are omitted here.

Vector splines and penalized likelihood

VGAMs employ *vector* smoothers in their estimation. One type of vector smoother is the *vector spline* which minimizes the quantity

$$\sum_{i=1}^{n} \{\boldsymbol{y}_i - \boldsymbol{f}(x_i)\}^T \boldsymbol{\Sigma}_i^{-1} \{\boldsymbol{y}_i - \boldsymbol{f}(x_i)\} \; + \sum_{j=1}^{M} \lambda_j \int_a^b \{f_j''(t)\}^2 \, dt \tag{12}$$

to scatterplot data $(x_i, \boldsymbol{y}_i, \boldsymbol{\Sigma}_i)$, $i = 1, \ldots, n$. Here, $\boldsymbol{\Sigma}_i$ are known symmetric and positive-definite error covariances, and $a < \min(x_1, \ldots, x_n)$ and $b > \max(x_1, \ldots, x_n)$. The first term of (12) measures lack of fit while the second term is a smoothing penalty. Equation (12) simplifies to an ordinary cubic smoothing spline when $M = 1$, (see, e.g., Green and Silverman, 1994). Each *component function* f_j has a non-negative smoothing parameter λ_j which operates as with an ordinary cubic spline.

Now consider the penalized likelihood

$$\ell - \frac{1}{2} \sum_{k=1}^{p} \sum_{j=1}^{M} \lambda_{(j)k} \int_{a_k}^{b_k} \{f_{(j)k}''(x_k)\}^2 \, dx_k. \tag{13}$$

Special cases of this quantity have appeared in a wide range of situations in the statistical literature to introduce cubic spline smoothing into a model's parameter estimation (for a basic overview of this roughness penalty approach see Green and Silverman, 1994). It transpires that the VGAM local scoring algorithm can be justified via maximizing the penalized likelihood (13) by vector spline smoothing (12) with $\mathbf{W}_i = \boldsymbol{\Sigma}_i^{-1}$.

From a practitioner's viewpoint the smoothness of a smoothing spline can be more conveniently controlled by specifying the degrees of freedom of the smooth rather than λ_j. Given a value of the degrees of freedom, the software can search for the λ_j producing this value. In general, the higher the degrees of freedom, the more wiggly the curve.

RR-VGLMs

Partition $\boldsymbol{x}$ into $(\boldsymbol{x}_1^T, \boldsymbol{x}_2^T)^T$ and $\mathbf{B} = (\mathbf{B}_1^T \; \mathbf{B}_2^T)^T$. In general, $\mathbf{B}$ is a dense matrix of full rank, i.e., rank $= \min(M, p)$. Thus there are $M \times p$ regression coefficients to estimate, and for some models and data sets this is "too" large.

One solution is based on a simple and elegant idea: replace $\mathbf{B}_2$ by a reduced-rank regression. This will cut down the number of regression coefficients enormously if the rank R is kept low. Ideally, the problem can be reduced down to one or two dimensions—a successful application of dimension reduction—and therefore can be plotted. The reduced-rank regression is applied to $\mathbf{B}_2$ because we want to make provision for some variables $\boldsymbol{x}_1$ that we want to leave alone, e.g., the intercept.

It transpires that *Reduced Rank VGLMs* (RR-VGLMs) are simply VGLMs where the constraint matrices are estimated. The modelling function `rrvglm()` calls an alternating algorithm which toggles between estimating two thin matrices $\mathbf{A}$ and $\mathbf{C}$, where $\mathbf{B}_2 = \mathbf{A}\,\mathbf{C}^T$.

Incidentally, special cases of RR-VGLMs have appeared in the literature. For example, a RR-multinomial logit model, or RR-MLM, is known as the *stereotype* model (Anderson, 1984). Another is Goodman (1981)'s RC model which is reduced-rank multivariate Poisson model. Note that the parallelism assumption of the proportional odds model (McCullagh and Nelder, 1989) can be thought of as a type of reduced rank regression where the constraint matrices are thin and known.

QRR-VGLMs and constrained ordination

RR-VGLMs form an optimal linear combination of the explanatory variables $\boldsymbol{x}_2$ and then fit a VGLM to these. Thus, in terms of (ecological) ordination, it performs a constrained linear ordination or CLO. Biotic responses are usually unimodal, and therefore this suggest fitting a quadratic on the $\boldsymbol{\eta}$ scale. This gives rise to the class of Quadratic RR-VGLMs, or QRR-VGLMs, so as to have constrained quadratic ordination or CQO. The result are bell-shaped curves/surfaces on axes defined in terms of gradients. The CQO approach here is statistically more rigorous than the popular canonical correspondence analysis (CCA) method. For more details see the references in Table 1.

Some user-oriented topics

Making the most of **VGAM** requires an understanding of the general framework described above plus a few other notions. Here are some of them.

Common arguments

VGAM family functions share a pool of common types of arguments, e.g., `exchangeable`, `nsimEIM`, `parallel`, `zero`. These are more convenient shortcuts for the argument `constraints` which accepts a named list of constraint matrices $\mathbf{H}_k$. For example, setting `parallel = TRUE` would constrain the coefficients $\beta_{(j)k}$ in (7) to be equal for all $j = 1, \ldots, M$, each separately for $k = 2, \ldots, p$. For some models the $\beta_{(j)1}$ are equal too. Another example is the `zero` argument which accepts a vector specifying which η_j is to be modelled as an intercept-only. Assigning it `NULL` means none. A last example is `nsimEIM`: some **VGAM** family functions use simulation to es-

timate the EIM and for these this argument controls the number of random variates generated.

Link functions

Most **VGAM** family functions allow a flexible choice of link functions to be assigned to each η_j. Table 3 lists some of them. Most **VGAM** family functions offer a link argument for each parameter plus an extra argument to pass in any value associated with the link function, e.g., an offset or power.

Random variates and EIMs

Many `dpqr`-functions exist to return the density, distribution function, quantile function and random generation of their respective **VGAM** family function. For example, there are the `[dpqr]sinmad()` functions corresponding to the Singh-Maddala distribution **VGAM** family function `sinmad()`.

Incidentally, if you are a developer or writer of models/distributions where scoring is natural, then it helps writers of future **VGAM** family functions to have expressions for the EIM. A good example is Kleiber and Kotz (2003) whose many distributions have been implemented in **VGAM**. If EIMs are untractable then algorithms for generating random variates and an expression for the score vector is the next best choice.

Smart prediction (for interest only)

It is well known among seasoned R and S-PLUS users that `lm()` and `glm()`-type models used to have prediction problems. **VGAM** currently implements "smart prediction" (written by T. Yee and T. Hastie) whereby the parameters of parameter dependent functions are saved on the object. This means that functions such as `scale(x)`, `poly(x)`, `bs(x)` and `ns(x)` will *always* correctly predict. However, most R users are unaware that other R modelling functions will not handle pathological cases such as `I(poly(x))` and `bs(scale(x))`. In S-PLUS even terms such as `poly(x)` and `scale(x)` will not predict properly without special measures.

Examples

Here are some examples giving a flavour of **VGAM** that I hope convey some of its breadth (at the expense of depth); `http://www.stat.auckland.ac.nz/~yee/VGAM` has more examples as well as some fledging documentation and the latest prerelease version of **VGAM**.

Example 1: Negative binomial regression

Consider a simple example involving simulated data where the dispersion parameter is a function of x. Two independent responses are constructed. Note that the k parameter corresponds to the `size` argument.

```
set.seed(123)
x = runif(n <- 500)
y1 = rnbinom(n, mu=exp(3+x), size=exp(1+x))
y2 = rnbinom(n, mu=exp(2-x), size=exp(2*x))
fit = vglm(cbind(y1,y2) ~ x,
           fam = negbinomial(zero=NULL))
```

Then

```
> coef(fit, matrix=TRUE)
            log(mu1) log(k1) log(mu2) log(k2)
(Intercept)   2.96   0.741     2.10  -0.119
x             1.03   1.436    -1.24   1.865
```

An edited summary shows

```
> summary(fit)
...
Coefficients:
               Value Std. Error t value
(Intercept):1  2.958     0.0534   55.39
(Intercept):2  0.741     0.1412    5.25
```

Table 3: Some **VGAM** link functions currently available.

Link	$g(\theta)$	**Range of** θ
`cauchit`	$\tan(\pi(\theta-\frac{1}{2}))$	$(0,1)$
`cloglog`	$\log_e\{-\log_e(1-\theta)\}$	$(0,1)$
`fisherz`	$\frac{1}{2}\log_e\{(1+\theta)/(1-\theta)\}$	$(-1,1)$
`fsqrt`	$\sqrt{2\theta}-\sqrt{2(1-\theta)}$	$(0,1)$
`identity`	θ	$(-\infty,\infty)$
`loge`	$\log_e(\theta)$	$(0,\infty)$
`logc`	$\log_e(1-\theta)$	$(-\infty,1)$
`logit`	$\log_e(\theta/(1-\theta))$	$(0,1)$
`logoff`	$\log_e(\theta+A)$	$(-A,\infty)$
`probit`	$\Phi^{-1}(\theta)$	$(0,1)$
`powl`	θ^p	$(0,\infty)$
`rhobit`	$\log_e\{(1+\theta)/(1-\theta)\}$	$(-1,1)$

```
(Intercept):3  2.097     0.0851   24.66
(Intercept):4 -0.119     0.1748   -0.68
x:1            1.027     0.0802   12.81
x:2            1.436     0.2508    5.73
x:3           -1.235     0.1380   -8.95
x:4            1.865     0.4159    4.48
```

so that all estimates are within two standard errors of their true values.

Example 2: Proportional odds model

Here we fit the POM on data from Section 5.6.2 of McCullagh and Nelder (1989).

```
data(pneumo)
pneumo = transform(pneumo,
                   let=log(exposure.time))
fit = vglm(cbind(normal,mild,severe) ~ let,
           cumulative(par=TRUE, rev=TRUE),
           data = pneumo)
```

Then

```
> coef(fit, matrix=TRUE)
            logit(P[Y>=2]) logit(P[Y>=3])
(Intercept)        -9.6761       -10.5817
let                 2.5968         2.5968
> constraints(fit)   # Constraint matrices
$`(Intercept)`
     [,1] [,2]
[1,]    1    0
[2,]    0    1

$let
     [,1]
[1,]    1
[2,]    1
```

The constraint matrices are the $\mathbf{H}_k$ in (10) and (11). The estimated variance-covariance matrix, and some of the fitted values and prior weights, are

```
>  vcov(fit)
            (Intercept):1 (Intercept):2    let
(Intercept):1     1.753         1.772 -0.502
(Intercept):2     1.772         1.810 -0.509
let              -0.502        -0.509  0.145
> cbind(fitted(fit),
        weights(fit, type="prior"))[1:4,]
  normal    mild  severe
1  0.994 0.00356 0.00243 98
2  0.934 0.03843 0.02794 54
3  0.847 0.08509 0.06821 43
4  0.745 0.13364 0.12181 48
```

Example 3: constraint matrices

How can we estimate θ given a random sample of n observations from a $N(\mu = \theta, \sigma = \theta)$ distribution? One way is to use the **VGAM** family function `normal1()` with the constraint that the mean and standard deviation are equal. Its default is $\eta_1 = \mu$ and $\eta_2 = \log \sigma$ but we can use the identity link function for σ and set $\mathbf{H}_1 = (1, 1)^T$. Suppose $n = 100$ and $\theta = 10$. Then we can use

```
set.seed(123)
theta = 10
y = rnorm(n <- 100, mean = theta, sd = theta)
clist = list("(Intercept)" = rbind(1, 1))
fit = vglm(y ~ 1, normal1(lsd="identity"),
           constraints = clist)
```

Then we get $\widehat{\theta} = 9.7504$ from

```
> coef(fit, matrix = TRUE)
               mean     sd
(Intercept) 9.7504 9.7504
```

Consider a similar problem but from $N(\mu = \theta, \sigma^2 = \theta)$. To estimate θ make use of $\log \mu = \log \theta = 2 \log \sigma$ so that

```
set.seed(123)
y2 = rnorm(n, theta, sd = sqrt(theta))
clist2 = list("(Intercept)" = rbind(1, 0.5))
fit2 = vglm(y2 ~ 1, normal1(lmean="loge"),
            constraints = clist2)
```

Then we get $\widehat{\theta} = 10.191$ from

```
> (cfit2 <- coef(fit2, matrix = TRUE))
            log(mean) log(sd)
(Intercept)    2.3215  1.1608
> exp(cfit2[1,"log(mean)"])
[1] 10.191
```

It is left to the reader as an exercise to figure out how to estimate θ from a random sample from $N(\mu = \theta, \sigma = 1 + e^{-\theta})$, for $\theta = 1$, say.

Example 4: mixture models

The 2008 World Fly Fishing Championships (WFFC) was held in New Zealand a few months ago. Permission to access and distribute the data was kindly given, and here we briefly look at the length of fish caught in Lake Rotoaira during the competition. A histogram of the data is given in Figure 1(a). The data looks possibly bimodal, so let's see how well a mixture of two normal distributions fits the data. This distribution is usually estimated by the EM algorithm but `mix2normal1()` uses simulated Fisher scoring.

```
data(wffc)
fit.roto = vglm(length/10 ~ 1, data=wffc,
    mix2normal1(imu2=45, imu1=25, ESD=FALSE),
    subset = water=="Rotoaira")
```

Here, we convert mm to cm. The estimation was helped by passing in some initial values, and we could have constrained the two standard deviations to be equal for parsimony. Now

```
> coef(fit.roto, matrix = TRUE)
            logit(phi)    mu1 log(sd1)
(Intercept)    -1.8990 25.671   1.4676
               mu2 log(sd2)
(Intercept) 46.473   1.6473
> Coef(fit.roto)
    phi      mu1      sd1      mu2      sd2
 0.1302 25.6706  4.3388 46.4733  5.1928
```

The resulting density is overlaid on the histogram in Figure 1(b). The LHS normal distribution doesn't look particularly convincing and would benefit from further investigation.

Finally, for illustration's sake, let's try the `untransform` argument of `vcov()`. Under limited conditions it applies the delta method to get at the untransformed parameters. The use of `vcov()` here is dubious (why?); its (purported?) standard errors are

```
> round(sqrt(diag(vcov(fit.roto,
        untransform=TRUE))), dig=3)
  phi   mu1   sd1   mu2   sd2
0.026 1.071 0.839 0.422 0.319
```

A more detailed analysis of the data is given in Yee (2008). The 2008 WFFC was a successful and enjoyable event, albeit we got pipped by the Czechs.

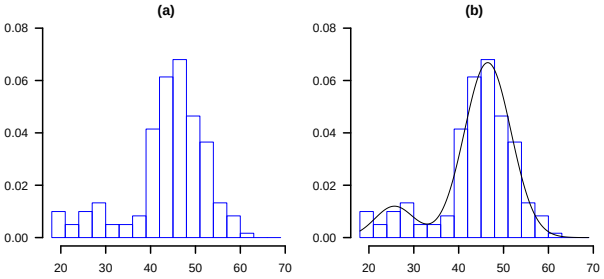


Figure 1: (a) Histogram of fish lengths (cm) at Lake Rotoaira. (b) Overlaid with the estimated mixture of two normal density functions. The sample size is 201.

Example 5: quantile & expectile regression

Over recent years quantile regression has become an important tool with a wide range of applications. The breadth of the VGLM/VGAM framework is illustrated by **VGAM** implementing *three* subclasses of quantile regression methods. We will only mention the two most popular subclasses here.

The first is of the LMS-type where a transformation of the response (e.g., Box-Cox) to some parametric distribution (e.g., standard normal) enables quantiles to be estimated on the transformed scale and then back-transformed to the original scale. In particular the popular Cole and Green (1992) method falls in this subclass. More details are in Yee (2004b).

The second is via expectile regression. Expectiles are almost as interpretable as quantiles because, given $X = x$, the percentile $\xi_\tau(x)$ specifies the position below which $100\tau\%$ of the (probability) mass of Y lies; while the expectile $\mu_\omega(x)$ determines, again given $X = x$, the point such that $100\omega\%$ of the mean distance between it and Y comes from the mass below it. The 0.5-expectile is the mean while the 0.5-quantile is the median. Expectile regression can be used to perform quantile regression by choosing ω that gives the desired τ. There are theoretical reasons why this is justified.

For normally distributed responses, expectile regression is based on *asymmetric least squares* (ALS) estimation, a variant of ordinary LS estimation. ALS estimation was generalized to *asymmetric maximum likelihood* (AML) estimation for members in the exponential family by Efron (1992). An example of this is the AML Poisson family which is illustrated in Figure 2. The data were generated by

```
set.seed(123)
alldat = data.frame(x=sort(runif(n<-500))+.2)
mymu = function(x)
    exp(-2 + 6*sin(2*x-0.2) / (x+0.5)^2)
alldat = transform(alldat,
                   y = rpois(n,mymu(x)))
```

Through trial and error we use

```
fit = vgam(y ~ s(x), data = alldat,
           amlpoisson(w=c(0.11, 0.9, 5.5)))
```

because the desired $\boldsymbol{\tau} = (0.25, 0.5, 0.75)^T$ obtained by

```
>  fit@extra[["percentile"]]
w.aml=0.11  w.aml=0.9  w.aml=5.5
      24.6       50.2       75.0
```

is to sufficient accuracy. Then Figure 2 was obtained by

```
with(alldat, plot(x, y, col="darkgreen"))
with(alldat, matlines(x, fitted(fit),
                      col="blue", lty=1))
```

If `parallel=TRUE` was set then this would avoid the embarrasing crossing quantile problem.

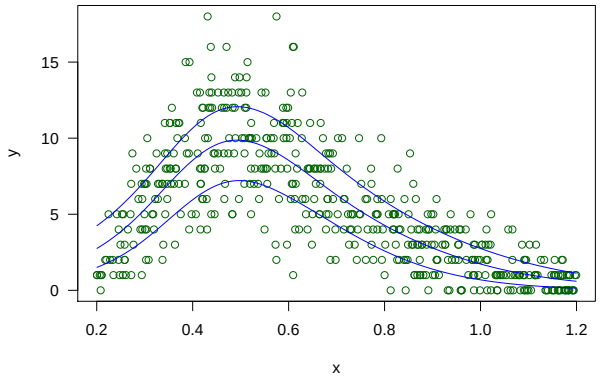


Figure 2: AML Poisson model fitted to the simulated Poisson data. The fitted expectiles correspond approximately to $\boldsymbol{\tau} = (0.25, 0.5, 0.75)^T$ and were fitted by smoothing splines.

Example 6: the bivariate logistic odds-ratio model

We consider the coalminers data of Table 6.6 of McCullagh and Nelder (1989). Briefly, 18282 working coalminers were classified by the presence of Y_1 = breathlessness and Y_2 = wheeze. The explanatory variable x = age was available. We fit a nonparametric bivariate logistic odds-ratio model

$$\begin{aligned} \eta_1 &= \text{logit}\, P(Y_1 = 1|x) = \beta_{(1)1} + f_{(1)1}(x), && (14)\\ \eta_2 &= \text{logit}\, P(Y_2 = 1|x) = \beta_{(2)1} + f_{(2)1}(x), && (15)\\ \eta_3 &= \log\, \psi = \beta_{(3)1} + f_{(3)1}(x), && (16) \end{aligned}$$

where $\psi = p_{11}p_{00}/(p_{10}p_{01})$ is the odds ratio, $p_{jk} = P(Y_1 = j, Y_2 = k|x)$. It is a good idea to afford $f_{(3)1}(x)$ less flexibility, and so we use

```
data(coalminers)
fit.coal = vgam(cbind(nBnW,nBW,BnW,BW) ~
      s(age, df=c(4,4,3)),
      binom2.or(zero=NULL), coalminers)
mycols = c("blue","darkgreen","purple")
plot(fit.coal, se=TRUE, lcol=mycols,
     scol=mycols, overlay=TRUE)
```

to give Figure 3. It appears that the log odds ratio could be modelled linearly.

Were an exchangeable error structure true we would expect both functions $\widehat{f}_{(1)1}(x)$ and $\widehat{f}_{(2)1}(x)$ to be on top of each other. However it is evident that this is not so, relative to the standard error bands. Possibly a quadratic function for each is suitable.

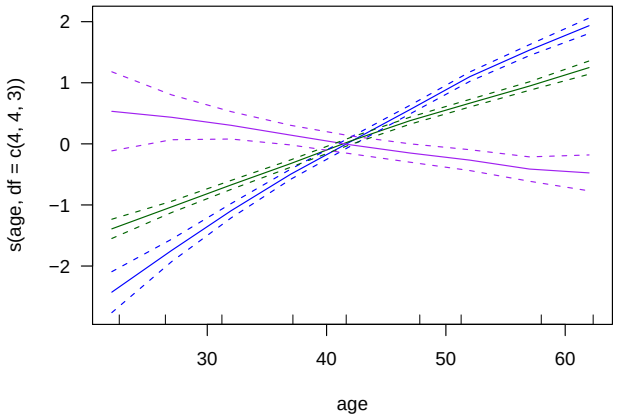


Figure 3: Bivariate logistic odds-ratio model fitted to the coalminers data. This is the VGAM (14)–(16) with centered components overlaid: $\widehat{f}_{(1)1}(x)$ in blue (steepest curve), $\widehat{f}_{(2)1}(x)$ in green, $\widehat{f}_{(3)1}(x)$ in purple (downward curve). The dashed curves are pointwise ± 2 standard error bands

Example 7: the Gumbel model

Central to classical extreme value theory is the generalized extreme value (GEV) distribution of which the Gumbel is a special case when $\xi = 0$. Suppose the maxima is Gumbel and let $Y_{(1)}, \ldots, Y_{(r)}$ be the r largest observations such that $Y_{(1)} \geq \cdots \geq Y_{(r)}$. The joint distribution of

$$\left(\frac{Y_{(1)} - b_n}{a_n}, \ldots, \frac{Y_{(r)} - b_n}{a_n}\right)^T$$

has, for large n, a limiting distribution with density $f(y_{(1)}, \ldots, y_{(r)}; \mu, \sigma) =$

$$\sigma^{-r} \exp\left\{-\exp\left(-\frac{y_{(r)} - \mu}{\sigma}\right) - \sum_{j=1}^{r}\left(\frac{y_{(j)} - \mu}{\sigma}\right)\right\},$$

for $y_{(1)} \geq \cdots \geq y_{(r)}$. Upon taking logarithms, one can treat this as an approximate log-likelihood.

The **VGAM** family function `gumbel()` fits this model. Its default is $\boldsymbol{\eta}(x) = (\mu(x), \log\, \sigma(x))^T$. We apply the block-Gumbel model to the well-known Venice sea levels data where the 10 highest sea levels (in cm) for each year for the years x = 1931 to 1981 were recorded. Note that only the top 6 values are available in 1935. For this reason let's try using the top 5 values. We have

```
data(venice)
fit=vglm(cbind(r1,r2,r3,r4,r5)~I(year-1930),
         gumbel(R=365, mpv=TRUE, zero=2,
                lscale="identity"),
         data = venice)
```

giving

```
> coef(fit, mat=TRUE)
                location scale
(Intercept)      104.246  12.8
I(year - 1930)     0.458   0.0
```

This agrees with Smith (1986).

Now let's look at the first 10 order statistics and introduce some splines such as Rosen and Cohen (1996). As they do, let's assume no serial correlation.

```
y=as.matrix(venice[,paste("r",1:10,sep="")])
fit1 = vgam(y ~ s(year, df=3),
            gumbel(R=365, mpv=TRUE),
            data=venice, na.action=na.pass)
```

A plot of the fitted quantiles

```
mycols = 1:3
qtplot(fit1, mpv=TRUE, lcol=mycols,
       tcol=mycols, las=1, llwd=2,
       ylab="Venice sea levels (cm)",
       pcol="blue", tadj=0.1, bty="l")
```

produces Figure 4. The curves suggest a possible inflection point in the mid-1960s. The *median predicted value* (MPV) for a particular year is the value that the maximum of the year has an even chance of exceeding. As a check, the plot shows about 26 values exceeding the MPV curve—this is about half of the 51 year values.

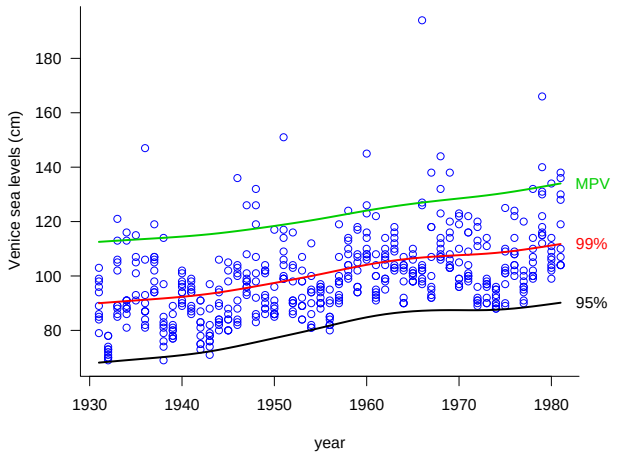


Figure 4: Block-Gumbel model fitted to the Venice sea levels data, smoothing splines are used.

More details about the application of VGLMs and VGAMs to extreme value data analysis are given in Yee and Stephenson (2007).

Example 8: nonlinear regression

The VGLM framework enables a Gauss-Newton-like algorithm to be performed in order to solve the nonlinear regression model

$$Y_i = f(\boldsymbol{u}_i; \boldsymbol{\theta}) + \varepsilon_i, \quad i = 1, \ldots, n, \qquad (17)$$

where $\boldsymbol{\theta}$ is an M-vector of unknown parameters and the ε_i are assumed to be i.i.d. $N(0, \sigma^2)$. Here, it is more convenient to use $\boldsymbol{u}$ to denote the regressors rather than the usual $\boldsymbol{x}$. An example of (17) is the Michaelis-Menten model

$$Y = \frac{\theta_1 u}{\theta_2 + u} + \varepsilon$$

and this is implemented in the **VGAM** family function `micmen()`. One advantage **VGAM** has over `nls()` for fitting this model is that `micmen()` is self-starting, i.e., initial values are automatically chosen.

Here is the Michaelis-Menten model fitted to some enzyme velocity and substrate concentration data (see Figure 5).

```
data(enzyme)
fit = vglm(velocity ~ 1, micmen, enzyme,
           form2 = ~ conc - 1)
with(enzyme, plot(conc, velocity, las=1,
                  xlab="concentration",
                  ylim=c(0,max(velocity)),
                  xlim=c(0,max(conc))))
with(enzyme, points(conc, fitted(fit),
                    col="red", pch="+"))

U = with(enzyme, max(conc))
newenzyme = data.frame(conc=seq(0,U,len=200))
fv = predict(fit, newenzyme, type="response")
with(newenzyme, lines(conc, fv, col="blue"))
```

The result is Figure 5.

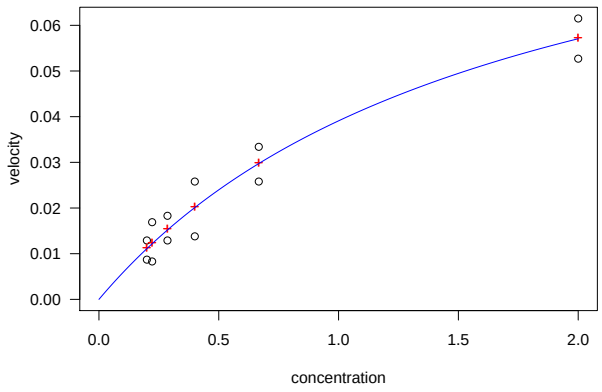


Figure 5: Michaelis-Menten model fitted to some enzyme data.

Example 9: constrained ordination

Ecologists frequently perform an ordination in order to see how multi-species and environmental data are related. An ordination enables one to display these data types simultaneously in an ordination plot.

Several methods have been developed of which one of the most common is canonical correspondence analysis (CCA). It was developed under the restrictive assumptions of a *species packing model* which stipulates that species have equal tolerances (a measure of niche width), equal maxima (a measure of abundance), and optima (optimal environments) that are uniformly distributed over the range of the gradient, which are linear combinations of the environmental variables (also known as latent variables).

CCA is a heuristic approximation to constrained quadratic ordination (CQO). The function `cqo()` implements CQO, meaning it assumes each species has a symmetric bell-shaped response with respect to underlying gradients. CQO does not necessarily make any of the species packing model assumptions.

A simple CQO model is where the responses are Poisson counts and there is one gradient. Then the model can be written like a Poisson regression

$$\log \mu_s = \beta_{(s)1} + \beta_{(s)2}\, \nu + \beta_{(s)3}\, \nu^2 \qquad (18)$$

$$= \alpha_s - \frac{1}{2}\left(\frac{\nu - u_s}{t_s}\right)^2. \qquad (19)$$

where $\nu = \boldsymbol{c}^T \boldsymbol{x}_2$ is the latent variable, and s denotes the species ($s = 1, \ldots, S$). The curves are unimodal provided $\beta_{(s)3}$ are negative. The second parameterization (19) has direct ecological interpretation: u_s is the species' optimum or *species score*, t_s is the tolerance, and $\exp(\alpha_s)$ is the maximum.

Here is an example using some hunting spiders data. The responses are 12 species' numbers trapped over a 60 week period and there are 6 environmental variables measured at 28 sites. The $\boldsymbol{x}$ are standardized prior to fitting.

```
data(hspider)
hspider[,1:6] = scale(hspider[,1:6])
p1 = cqo(cbind(Alopacce, Alopcune, Alopfabr,
               Arctlute, Arctperi, Auloalbi,
               Pardlugu, Pardmont, Pardnigr,
               Pardpull, Trocterr, Zoraspin)
       ~ WaterCon + BareSand + FallTwig +
         CoveMoss + CoveHerb + ReflLux,
         fam = poissonff, data = hspider,
         Crow1posit = FALSE, ITol = FALSE)
```

Then the fitted coefficients of (19) can be seen in

```
> coef(p1)

C matrix (constrained/canonical coefficients)
             lv
WaterCon -0.233
BareSand  0.513
FallTwig -0.603
CoveMoss  0.211
CoveHerb -0.340
ReflLux   0.798

...

Optima and maxima
         Optimum Maximum
Alopacce   1.679   19.29
Alopcune  -0.334   18.37
Alopfabr   2.844   13.03
Arctlute  -0.644    6.16
Arctperi   3.906   14.42
Auloalbi  -0.586   19.22
Pardlugu      NA      NA
Pardmont   0.714   48.60
Pardnigr  -0.530   87.80
Pardpull  -0.419  110.32
Trocterr  -0.683  102.27
Zoraspin  -0.742   27.24

Tolerance
            lv
Alopacce 1.000
Alopcune 0.849
Alopfabr 1.048
Arctlute 0.474
Arctperi 0.841
Auloalbi 0.719
Pardlugu    NA
Pardmont 0.945
Pardnigr 0.529
Pardpull 0.597
Trocterr 0.932
Zoraspin 0.708
```

All but one species have fitted bell-shaped curves. The $\widehat{\nu}$ can be interpreted as a moisture gradient. The curves may be plotted with

```
S = ncol(p1@y) # Number of species
clr = (1:(S+1))[-7] # omits yellow
persp(p1, col=clr, llty=1:S, llwd=2, las=1)
legend("topright", legend=colnames(p1@y),
       col=clr, bty="n", lty=1:S, lwd=2)
```

and this results in Figure 6.

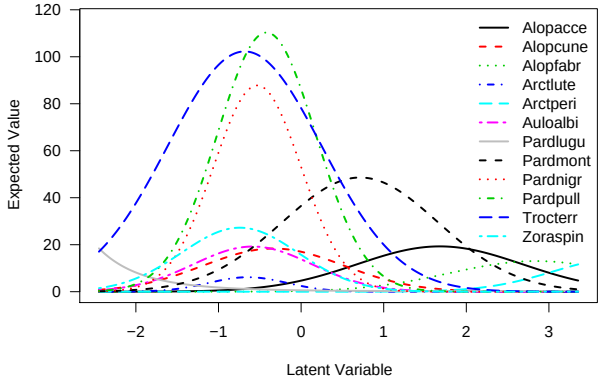


Figure 6: CQO Poisson model, with unequal tolerances, fitted to the hunting spiders data.

Currently `cqo()` will handle presence/absence data via `family = binomialff`. Altogether the present `cqo()` function is a first stab at ordination based on sound regression techniques. Since it fits by maximum likelihood estimation it is likely to be sensitive to outliers. The method requires very clear unimodal responses amongst the species, hence the $\boldsymbol{x}$ needs to be collected over a broad range of environments. All this means that it may be of limited use to 'real' biological data where messy data is common and robustness to outliers is needed.

Summary

Being a large project, **VGAM** is far from completion. Currently some of the internals are being rewritten and a monograph is in the making. Its continual development means changes to some details presented here may occur in the future. The usual maintenance, consisting of bug fixing and implementing improvements, applies.

In summary, the contributed R package **VGAM** is purposely general and computes the maximum likelihood estimates of many types of models and distributions. It comes not only with the capability to do a lot of things but with a large and unified framework that is flexible and easily understood. It is hoped that the package will be useful to many statistical practitioners.

Bibliography

A. Agresti. *Categorical Data Analysis*. Wiley, New York, second edition, 2002.

J. A. Anderson. Regression and ordered categorical variables (with discussion). *Journal of the Royal Sta-*

tistical Society, Series B, Methodological, 46(1):1–30, 1984.

J. M. Chambers. *Programming with Data: A Guide to the S Language*. Springer, New York, 1998.

T. J. Cole and P. J. Green. Smoothing reference centile curves: The LMS method and penalized likelihood. *Statistics in Medicine*, 11(10):1305–1319, 1992.

B. Efron. Poisson overdispersion estimates based on the method of asymmetric maximum likelihood. *Journal of the American Statistical Association*, 87(417):98–107, 1992.

B. Efron and D. V. Hinkley. Assessing the accuracy of the maximum likelihood estimator: Observed versus expected Fisher information. *Biometrika*, 65(3): 457–481, 1978.

L. A. Goodman. Association models and canonical correlation in the analysis of cross-classifications having ordered categories. *Journal of the American Statistical Association*, 76(374):320–334, 1981.

P. J. Green and B. W. Silverman. *Nonparametric Regression and Generalized Linear Models: A Roughness Penalty Approach*. Chapman & Hall, London, 1994.

T. J. Hastie and R. J. Tibshirani. *Generalized Additive Models*. Chapman & Hall, London, 1990.

C. Kleiber and S. Kotz. *Statistical Size Distributions in Economics and Actuarial Sciences*. Wiley-Interscience, Hoboken, NJ, USA, 2003.

P. McCullagh and J. A. Nelder. *Generalized Linear Models*. Chapman & Hall, London, second edition, 1989.

O. Rosen and A. Cohen. Extreme percentile regression. In W. Härdle and M. G. Schimek, editors, *Statistical Theory and Computational Aspects of Smoothing: Proceedings of the COMPSTAT '94 Satellite Meeting held in Semmering, Austria, 27–28 August 1994*, pages 200–214, Heidelberg, 1996. Physica-Verlag.

R. L. Smith. Extreme value theory based on the r largest annual events. *Journal of Hydrology*, 86(1–2):27–43, 1986.

T. W. Yee. A new technique for maximum-likelihood canonical Gaussian ordination. *Ecological Monographs*, 74(4):685–701, 2004a.

T. W. Yee. Quantile regression via vector generalized additive models. *Statistics in Medicine*, 23(14):2295–2315, 2004b.

T. W. Yee. Constrained additive ordination. *Ecology*, 87(1):203–213, 2006.

T. W. Yee. Vector generalized linear and additive models, with applications to the 2008 World Fly Fishing Championships. *In preparation*, 2008.

T. W. Yee and T. J. Hastie. Reduced-rank vector generalized linear models. *Statistical Modelling*, 3(1): 15–41, 2003.

T. W. Yee and A. G. Stephenson. Vector generalized linear and additive extreme value models. *Extremes*, 10(1–2):1–19, 2007.

T. W. Yee and C. J. Wild. Vector generalized additive models. *Journal of the Royal Statistical Society, Series B, Methodological*, 58(3):481–493, 1996.

Thomas W. Yee
Department of Statistics
University of Auckland
New Zealand
t.yee@auckland.ac.nz

Comparing Non-Identical Objects

Introducing the compare Package

by Paul Murrell

The **compare** package provides functions for comparing two R objects for equality, while allowing for a range of "minor" differences. Objects may be reordered, rounded, or resized, they may have names or attributes removed, or they may even be coerced to a new class if necessary in order to achieve equality.

The results of comparisons report not just whether the objects are the same, but also include a record of any modifications that were performed.

This package was developed for the purpose of partially automating the marking of coursework involving R code submissions, so functions are also provided to convert the results of comparisons into numeric grades and to provide feedback for students.

Motivation

STATS 220 is a second year university course run by the Department of Statistics at the University of Auckland.[1] The course covers a range of "Data Technologies", including HTML, XML, databases, SQL, and, as a general purpose data processing tool, R.

In addition to larger assignments, students in the course must complete short exercises in weekly computer labs.

For the R section of the course, students must write short pieces of R code to produce specific R objects. Figure 1 shows two examples of basic, introductory exercises.

The students submit their answers to the exercises as a file containing R code, which means that it is possible to recreate their answers by calling `source()` on the submitted files.

At this point, the R objects generated by the students' code can be compared with a set of model R objects in order to establish whether the students' answers are correct.

How this comparison occurs is the focus of this article.

Black and white comparisons

The simplest and most strict test for equality between two objects in the base R system (R Development Core Team, 2008) is provided by the function `identical()`. This returns `TRUE` if the two objects are *exactly* the same, otherwise it returns `FALSE`.

The problem with this function is that it is very strict indeed and will fail for objects that are, for all practical purposes, the same. The classic example is the comparison of two real (floating-point) values, as demonstrated in the following code, where differences can arise simply due to the limitations of how numbers are represented in computer memory (see R FAQ 7.31, Hornik, 2008).

```
> identical(0.3 - 0.2, 0.1)

[1] FALSE
```

Using the function to test for equality would clearly be unreasonably harsh when marking any student answer that involves calculating a numeric result.

The `identical()` function, by itself, is not sufficient for comparing student answers with model answers.

Shades of grey

The recommended solution to the problem mentioned above of comparing two floating-point values is to use the `all.equal()` function. This function allows for "insignificant" differences between numeric values, as shown below.

```
> all.equal(0.3 - 0.2, 0.1)

[1] TRUE
```

This makes `all.equal()` a much more appropriate function for comparing student answers with model answers.

What is less well-known about the `all.equal()` function is that it also works for comparing other sorts of R objects, besides numeric vectors, *and* that it does more than just report equality between two objects.

If the objects being compared have differences, then `all.equal()` does not simply return `FALSE`. Instead, it returns a character vector containing messages that describe the differences between the objects. The following code gives a simple example, where `all.equal()` reports that the two character vectors have different lengths, and that, of the two pairs of strings that can be compared, one pair of strings does not match.

```
> all.equal(c("a", "b", "c"), c("a", "B"))

[1] "Lengths (3, 2) differ (string compare
                                    on first 2)"
[2] "1 string mismatch"
```

This feature is actually very useful for marking student work. Information about whether a student's answer is correct is useful for determining a raw mark, but it is also useful to have information about what the student did wrong. This information

[1] `http://www.stat.auckland.ac.nz/courses/stage2/#STATS220`

1. Write R code to create the three **vectors** and the **factor** shown below, with names `id`, `age`, `edu`, and `class`.

   You should end up with objects that look like this:

```
> id
[1] 1 2 3 4 5 6

> age
[1] 30 32 28 39 20 25

> edu
[1] 0 0 0 0 0 0

> class
[1] poor   poor   poor   middle
[5] middle middle
Levels: middle poor
```

2. Combine the objects from Question 1 together to make a **data frame** called `IndianMothers`.

   You should end up with an object that looks like this:

```
> IndianMothers
  id age edu  class
1  1  30   0   poor
2  2  32   0   poor
3  3  28   0   poor
4  4  39   0 middle
5  5  20   0 middle
6  6  25   0 middle
```

Figure 1: Two simple examples of the exercises that STATS 220 students are asked to perform.

can be used as the basis for assigning partial marks for an answer that is close to the correct answer, and for providing feedback to the student about where marks were lost.

The `all.equal()` function has some useful features that make it a helpful tool for comparing student answers with model answers. However, there is an approach that can perform better than this.

The `all.equal()` function looks for equality between two objects and, if that fails, provides information about the sort of differences that exist. An alternative approach, when two objects are not equal, is to try to *transform* the objects to make them equal, and report on which transformations were necessary in order to achieve equality.

As an example of the difference between these approaches, consider the two objects below: a character vector and a factor.

```
> obj1 <- c("a", "a", "b", "c")
> obj1

[1] "a" "a" "b" "c"

> obj2 <- factor(obj1)
> obj2

[1] a a b c
Levels: a b c
```

The `all.equal()` function reports that these objects are different because they differ in terms of their fundamental mode—one has attributes and the other does not—and because each object is of a different class.

```
> all.equal(obj1, obj2)
[1] "Modes: character, numeric"
[2] "Attributes: < target is NULL, current
                                    is list >"
[3] "target is character, current is factor"
```

The alternative approach would be to allow various transformations of the objects to see if they can be transformed to be the same. The following code shows this approach, which reports that the objects are equal, if the second one is coerced from a factor to a character vector. This is more information than was provided by `all.equal()` and, in the particular case of comparing student answers to model answers, it tells us a lot about how close the student got to the right answer.

```
> library(compare)
> compare(obj1, obj2, allowAll=TRUE)

TRUE
  coerced from <factor> to <character>
```

Another limitation of `all.equal()` is that it does not report on some other possible differences between objects. For example, it is possible for a student to have the correct values for an R object, but have the values in the wrong order. Another common mistake is to get the case wrong in a set of string values (e.g., in a character vector or in the `names` attribute of an object).

In summary, while `all.equal()` provides some desirable features for comparing student answers to model answers, we can do better by allowing for a wider range of differences between objects and by taking a different approach that attempts to transform the student answer to be the same as the model answer, if at all possible, while reporting which transformations were necessary.

The remainder of this article describes the **compare** package, which provides functions for producing these sorts of comparisons.

The `compare()` function

The main function in the **compare** package is the `compare()` function. This function checks whether two objects are the same and, if they are not, carries out various transformations on the objects and checks them again to see if they are the same after they have been transformed.

By default, `compare()` only succeeds if the two objects are identical (using the `identical()` function) *or* the two objects are numeric and they are equal (according to `all.equal()`). If the objects are not the same, no transformations of the objects are considered. In other words, by default, `compare()` is simply a convenience wrapper for `identical()` and `all.equal()`. As a simple example, the following comparison takes account of the fact that the values being compared are numeric and uses `all.equal()` rather than `identical()`.

```
> compare(0.3 - 0.2, 0.1)

TRUE
```

Transformations

The more interesting uses of `compare()` involve specifying one or more of the arguments that allow transformations of the objects that are being compared. For example, the `coerce` argument specifies that the second argument may be coerced to the class of the first argument. This allows for more flexible comparisons such as between a factor and a character vector.

```
> compare(obj1, obj2, coerce=TRUE)

TRUE
  coerced from <factor> to <character>
```

It is important to note that there is a definite order to the objects; the *model* object is given first and the *comparison* object is given second. Transformations attempt to make the comparison object like the model object, though in a number of cases (e.g., when ignoring the case of strings) the model object may also be transformed. In the example above, the comparison object has been coerced to be the same class as the model object. The following code demonstrates the effect of reversing the order of the objects in the comparison. Now the character vector is being coerced to a factor.

```
> compare(obj2, obj1, coerce=TRUE)

TRUE
  coerced from <character> to <factor>
```

Of course, transforming an object is not guaranteed to produce identical objects if the original objects are genuinely different.

```
> compare(obj1, obj2[1:3], coerce=TRUE)

FALSE
  coerced from <factor> to <character>
```

Notice, however, that even though the comparison failed, the result still reports the transformation that was attempted. This result indicates that the comparison object was converted from a factor (to a character vector), but it *still* did not end up being the same as the model object.

A number of other transformations are available in addition to coercion. For example, differences in length, like in the last case, can also be ignored.

```
> compare(obj1, obj2[1:3],
+         shorten=TRUE, coerce=TRUE)

TRUE
  coerced from <factor> to <character>
  shortened model
```

It is also possible to allow values to be sorted, or rounded, or to convert all character values to upper case (i.e., ignore the case of strings).

Table 1 provides a complete list of the transformations that are currently allowed (in version 0.2 of **compare**) and the arguments that are used to enable them.

A further argument to the `compare()` function, `allowAll`, controls the default setting for most of these transformations, so specifying `allowAll=TRUE` is a quick way of enabling all possible transformations. Specific transformations can still be *excluded* by explicitly setting the appropriate argument to `FALSE`.

The `equal` argument is a bit of a special case because it is `TRUE` by default, whereas almost all others are `FALSE`. The `equal` argument is also especially influential because objects are compared after every transformation and this argument controls what sort of comparison takes place. Objects are always compared using `identical()` first, which will only succeed if the objects have exactly the same representation in memory. If the test using `identical()` fails and `equal=TRUE`, then a more lenient comparison is also performed. By default, this just means that numeric values are compared using `all.equal()`, but various other arguments can extend this to allow things like differences in case for character values (see the asterisked arguments in Table 1).

The `round` argument is also special because it always defaults to `FALSE`, even if `allowAll=TRUE`. This means that the `round` argument must be specified explicitly in order to enable rounding. The default is set up this way because the value of the `round` argument is either `FALSE` or an integer value specifying the number of decimal places to round to. For this argument, the value `TRUE` corresponds to rounding to zero decimal places.

Table 1: Arguments to the `compare()` function that control which transformations are attempted when comparing a model object to a comparison object.

Argument	Meaning
`equal`	Compare objects for "equality" as well as "identity" (e.g., use `all.equal()` if model object is numeric).
`coerce`	Allow coercion of comparison object to class of model object.
`shorten`	Allow either the model or the comparison to be shrunk so that the objects have the same "size".
`ignoreOrder`	Ignore the original order of the comparison and model objects; allow both comparison object and model object to be sorted.
`ignoreNameCase`	Ignore the case of the `names` attribute for both comparison and model objects; the `name` attributes for both objects are converted to upper case.
`ignoreNames`	Ignore any differences in the `names` attributes of the comparison and model objects; any `names` attributes are dropped.
`ignoreAttrs`	Ignore all attributes of both the comparison and model objects; all attributes are dropped.
`round`*	Allow numeric values to be rounded; either `FALSE` (the default), or an integer value giving the number of decimal places for rounding, or a function of one argument, e.g., `floor`.
`ignoreCase`*	Ignore the case of character vectors; both comparison and model are converted to upper case.
`trim`*	Ignore leading and trailing spaces in character vectors; leading and trailing spaces are trimmed from both comparison and model.
`ignoreLevelOrder`*	Ignore original order of levels of factor objects; the levels of the comparison object are sorted to the order of the levels of the model object.
`dropLevels`*	Ignore any unused levels in factors; unused levels are dropped from both comparison and model objects.
`ignoreDimOrder`	Ignore the order of dimensions in array, matrix, or table objects; the dimensions are reordered by name.
`ignoreColOrder`	Ignore the order of columns in data frame objects; the columns in the comparison object are reordered to match the model object.
`ignoreComponentOrder`	Ignore the order of components in a list object; the components are reordered by name.

*These transformations only occur if `equal=TRUE`

Finally, there is an additional argument `colsOnly` for comparing data frames. This argument controls whether transformations are only applied to columns (and not to rows). For example, by default, a data frame will only allow columns to be dropped, but not rows, if `shorten=TRUE`. Note, however, that `ignoreOrder` means ignore the order of *rows* for data frames and `ignoreColOrder` must be used to ignore the order of columns in comparisons involving data frames.

The `compareName()` function

The `compareName()` function offers a slight variation on the `compare()` function.

For this function, only the *name* of the comparison object is specified, rather than an explicit object. The advantage of this is that it allows for variations in case in the names of objects. For example, a student might create a variable called `indianMothers` rather than the desired `IndianMothers`. This case-insensitivity is enabled via the `ignore.case` argument.

Another advantage of this function is that it is possible to specify, via the `compEnv` argument, a particular environment to search within for the comparison object (rather than just the current workspace). This becomes useful when checking the answers from several students because each student's answers may be generated within a separate environment in order to avoid any interactions between code from different students.

The following code shows a simple demonstration of this function, where a comparison object is created within a temporary environment and the name of the comparison object is upper case when it should be lowercase.

```
> tempEnv <- new.env()
> with(tempEnv, X <- 1:10)
> compareName(1:10, "x", compEnv=tempEnv)

TRUE
  renamed object
```

Notice that, as with the transformations in `compare()`, the `compareName()` function records whether it needed to ignore the case of the name of the comparison object.

A pathological example

This section shows a manufactured example that demonstrates some of the flexibility of the `compare()` function.

We will compare two data frames that have a number of simple differences. The model object is a data frame with three columns: a numeric vector, a character vector, and a factor.

```
> model <-
+     data.frame(x=1:26,
+                y=letters,
+                z=factor(letters),
+                row.names=letters,
+                stringsAsFactors=FALSE)
```

The comparison object contains essentially the same information, except that there is an extra column, the column names are uppercase rather than lowercase, the columns are in a different order, the `y` variable is a factor rather than a character vector, and the `z` variable is a character variable rather than a factor. The `y` variable and the row names are also uppercase rather than lowercase.

```
> comparison <-
+     data.frame(W=26:1,
+                Z=letters,
+                Y=factor(LETTERS),
+                X=1:26,
+                row.names=LETTERS,
+                stringsAsFactors=FALSE)
```

The `compare()` function can detect that these two objects are essentially the same as long as we reorder the columns (ignoring the case of the column names), coerce the `y` and `z` variables, drop the extra variable, ignore the case of the `y` variable, and ignore the case of the row names.

```
> compare(model, comparison, allowAll=TRUE)

TRUE
  renamed
  reordered columns
  [Y] coerced from <factor> to <character>
  [Z] coerced from <character> to <factor>
  shortened comparison
  [Y] ignored case
  renamed rows
```

Notice that we have used `allowAll=TRUE` to allow `compare()` to attempt all possible transformations at its disposal.

Comparing files of R code

Returning now to the original motivation for the **compare** package, the `compare()` function provides an excellent basis for determining not only whether a student's answers are correct, but also how much incorrect answers differ from the model answer.

As described earlier, submissions by students in the STATS 220 course consist of files of R code. Marking these submissions consists of using `source()` to run the code, then comparing the resulting objects with model answer objects. With approximately 100 students in the STATS 220 course, with weekly labs, and with multiple questions per lab, each of which may contain more than one R object, there is a reasonable marking burden. Consequently, there is a strong incentive to automate as much of the marking process as possible.

The `compareFile()` function

The `compareFile()` function can be used to run R code from a specific file and compare the results with a set of model answers. This function requires three pieces of information: the name of a file containing the "comparison code", which is run within a local environment, using `source()`, to generate the comparison values; a vector of "model names", which are the names of the objects that will be looked for in the local environment after the comparison code has been run; and the model answers, either as the name of a binary file to `load()`, or as the name of a file of R code to `source()`, or as a list object containing the ready-made model answer objects.

Any argument to `compare()` may also be included in the call.

Once the comparison code has been run, `compareName()` is called for each of the model names and the result is a list of `"comparison"` objects.

As a simple demonstration, consider the basic questions shown in Figure 1. The model names in this case are the following:

```
> modelNames <- c("id", "age",
+                 "edu", "class",
+                 "IndianMothers")
```

One student's submission for this exercise is in a file called `student1.R`, within a directory called `Examples`. The model answer is in a file called `model.R` in the same directory. We can evaluate this student's submission and compare it to the model answer with the following code:

```
> compareFile(file.path("Examples",
+                       "student1.R"),
+             modelNames,
+             file.path("Examples",
+                       "model.R"))

$id
TRUE

$age
TRUE

$edu
TRUE

$class
FALSE

$IndianMothers
FALSE
  object not found
```

This provides a strict check and shows that the student got the first three problems correct, but the last two wrong. In fact, the student's code completely failed to generate an object with the name `IndianMothers`.

We can provide extra arguments to allow transformations of the student's answers, as in the following code:

```
> compareFile(file.path("Examples",
+                       "student1.R"),
+             modelNames,
+             file.path("Examples",
+                       "model.R"),
+             allowAll=TRUE)


$id
TRUE

$age
TRUE

$edu
TRUE

$class
TRUE
  reordered levels

$IndianMothers
FALSE
  object not found
```

This shows that, although the student's answer for the `class` object was not perfect, it was pretty close; it just had the levels of the factor in the wrong order.

The `compareFiles()` function

The `compareFiles()` function builds on `compareFile()` by allowing a vector of comparison file names. This allows a whole set of student submissions to be tested at once. The result of this function is a list of lists of `"comparison"` objects and a special print method provides a simplified view of this result.

Continuing the example from above, the `Examples` directory contains submissions from a further four students. We can compare all of these submissions with the model answers and produce a summary of the results with a single call to `compareFiles()`. The appropriate code and output are shown in Figure 2.

The results show that most students got the first three problems correct. They had more trouble getting the fourth problem right, with one getting the factor levels in the wrong order and two others producing a character vector rather than a factor. Only one student, `student2`, got the final problem exactly right and only one other, `student4`, got essentially the right answer, though this student spelt the name of the object wrong.

```
> files <- list.files("Examples",
+                     pattern="^student[0-9]+[.]R$",
+                     full.names=TRUE)
> results <- compareFiles(files,
+                         modelNames,
+                         file.path("Examples", "model.R"),
+                         allowAll=TRUE,
+                         resultNames=gsub("Examples.|[.]R", "", files))
> results
```

```
         id    age   edu   class                                        IndianMothers
student1 TRUE  TRUE  TRUE  TRUE reordered levels                       FALSE object not found
student2 TRUE  TRUE  TRUE  TRUE                                        TRUE
student3 TRUE  TRUE  TRUE  TRUE coerced from <character> to <factor> FALSE object not found
student4 TRUE  TRUE  TRUE  TRUE coerced from <character> to <factor> TRUE renamed object
student5 TRUE  TRUE  TRUE  FALSE object not found                      FALSE object not found
```

Figure 2: Using the `compareFiles()` function to run R code from several files and compare the results to model objects. The result of this sort of comparison can easily get quite wide, so it is often useful to print the result with `options(width)` set to some large value and using a small font, as has been done here.

Assigning marks and giving feedback

The result returned by `compareFiles()` is a list of lists of comparison results, where each result is itself a list of information including whether two objects are the same and a record of how the objects were transformed during the comparison. This represents a wealth of information with which to assess the performance of students on a set of R exercises, but it can be a little unwieldly to deal with.

The **compare** package provides further functions that make it easier to deal with this information for the purpose of determining a final mark and for the purpose of providing comments for each student submission.

In order to determine a final mark, we use the `questionMarks()` function to specify which object names are involved in a particular question, to provide a maximum mark for the question, and to specify a set of rules that determine how many marks should be deducted for various deviations from the correct answers.

The `rule()` function is used to define a marking rule. It takes an object name, a number of marks to deduct if the comparison for that object is `FALSE`, plus any number of transformation rules. The latter are generated using the `transformRule()` function, which associates a regular expression with a number of marks to deduct. If the regular expression is matched in the record of transformations for a comparison, then the appropriate number of marks are deducted.

A simple example, based on the second question in Figure 1, is shown below. This specifies that the question only involves an object named `IndianMothers`, that there is a maximum mark of 1 for this question, and that 1 mark is deducted if the comparison is `FALSE`.

```
> q2 <-
+     questionMarks("IndianMothers",
+                   maxMark=1,
+                   rule("IndianMothers", 1))
```

The first question from Figure 1 provides a more complex example. In this case, there are four different objects involved and the maximum mark is 2. The rules below specify that any `FALSE` comparison drops a mark *and* that, for the comparison involving the object named `"class"`, a mark should also be deducted if coercion was necessary to get a `TRUE` result.

```
> q1 <-
+     questionMarks(
+         c("id", "age", "edu", "class"),
+         maxMark=2,
+         rule("id", 1),
+         rule("age", 1),
+         rule("edu", 1),
+         rule("class", 1,
+              transformRule("coerced", 1)))
```

Having set up this marking scheme, marks are generated using the `markQuestions()` function, as shown by the following code.

```
> markQuestions(results, q1, q2)
```

```
         id-age-edu-class IndianMothers
student1                2             0
student2                2             1
student3                1             0
student4                1             1
student5                1             0
```

For the first question, the third and fourth students lose a mark because of the coercion, and the fifth student loses a mark because he has not generated the required object.

A similar suite of functions are provided to associate comments, rather than mark deductions, with particular transformations. The following code provides a simple demonstration.

```
> q1comments <-
+     questionComments(
+         c("id", "age", "edu", "class"),
+         comments(
+             "class",
+             transformComment(
+                 "coerced",
+                 "'class' is a factor!")))
> commentQuestions(results, q1comments)

         id-age-edu-class
student1 ""
student2 ""
student3 "'class' is a factor!"
student4 "'class' is a factor!"
student5 ""
```

In this case, we have just generated feedback for the students who generated a character vector instead of the desired factor in Question 1 of the exercise.

Summary, discussion, and future directions

The **compare** package is based around the `compare()` function, which compares two objects for equality and, if they are not equal, attempts to transform the objects to make them equal. It reports whether the comparison succeeded overall and provides a record of the transformations that were attempted during the comparison.

Further functions are provided on top of the `compare()` function to facilitate marking exercises where students in a class submit R code in a file to create a set of R objects.

This article has given some basic demonstrations of the use of the **compare()** package for comparing objects and marking student submissions. The package could also be useful for the students themselves, both to check whether they have the correct answer and to provide feedback about how their answer differs from the model answer. More generally, the `compare()` function may have application wherever the `identical()` and `all.equal()` functions are currently in use. For example, it may be useful when debugging code and for performing regression tests as part of a quality control process.

Obvious extensions of the **compare** package include adding new transformations and providing comparison methods for other classes of objects. More details about how the package works and how these extensions might be developed are discussed in the vignette, "Fundamentals of the Compare Package", which is installed as part of the **compare** package.

Acknowledgements

Many thanks to the editors and anonymous reviewers for their useful comments and suggestions, on both this article and the **compare** package itself.

Bibliography

K. Hornik. The R FAQ, 2008. URL `http://CRAN.R-project.org/doc/FAQ/R-FAQ.html`. ISBN 3-900051-08-9.

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

Paul Murrell
Department of Statistics
University of Auckland
New Zealand
`p.murrell@auckland.ac.nz`

mvna: An R Package for the Nelson–Aalen Estimator in Multistate Models

by A. Allignol, J. Beyersmann and M. Schumacher

Introduction

The multivariate Nelson–Aalen estimator of cumulative transition hazards is the fundamental nonparametric estimator in event history analysis. However, there has not been a multivariate Nelson–Aalen R–package available, and the same appears to be true for other statistical softwares, like SAS or Stata. As the estimator is algebraically simple, its univariate entries are easily computed in R using the `survfit` function of the package **survival** (Therneau and Grambsch, 2001; Lumley, 2004), but other values returned by `survfit` may be meaningless in the multivariate setting and consequently misleading (Kalbfleisch and Prentice, 2002, Chapter VIII.2.4). In addition, the computations become cumbersome for even a moderate number of transition types, like 4 or 5.

For instance, time–dependent covariates may easily be incorporated into a proportional hazards analysis, but, in general, this approach will not result in a model for event probabilities any more (Kalbfleisch and Prentice, 2002, Chapter VI.3). The multivariate Nelson–Aalen estimate straightforwardly illustrates the analysis, while popular stratified Kaplan-Meier plots lack a probability interpretation (*e.g.,* Feuer et al. (1992); Beyersmann et al. (2006b)).

The article is organised as follows: First we describe the main features of the **mvna** package. Next we use the package to illustrate the analysis of a time–dependent covariate. We briefly discuss the relative merits of the available variance estimators, and conclude with a summary.

Package description

The **mvna** contains the following functions:

- `mvna`
- `xyplot.mvna`
- `print.mvna`
- `predict.mvna`

The main function, `mvna`, computes the Nelson–Aalen estimates at each of the observed event times, and the two variance estimators described in eq. (4.1.6) and (4.1.7) of Andersen et al. (1993). `mvna` takes as arguments a transition–oriented data.frame, where each row represents a transition:

```
 id from   to time
  1    0    2    4
  2    1    2   10
  3    1    2    2
  4    1    2   49
  5    1    0   36
  5    0    2   47
749    1    0   11
749    0 cens   22
```

`id` is the patient identification number, `from` is the state from which a transition occurs, `to` is the state to which a transition occurs and `time` is the transition time. Left–truncation, *e.g.,* due to delayed entry into the study cohort, can be handled replacing `time` with an `entry` and `exit` time within a state. One need to specify the state names, the possible transitions, and the censoring code. For example, patient 749 is right–censored at time 22.

`xyplot.mvna` plots the cumulative hazard estimates in a lattice plot (Sarkar, 2002), along with pointwise confidence intervals, with possible log or arcsin transformations to improve the approximation to the asymptotic distribution. The `predict.mvna` function gives Nelson–Aalen estimates at time points given by the user.

Two random samples of intensive care unit data are also included within the **mvna** package (Beyersmann et al., 2006a). One of these data sets focuses on the effect of pneumonia on admission on the hazard of discharge and death, respectively. The other one contains data to gauge the influence of ventilation (a time–dependent covariate) on the hazard of end of hospital stay (combined endpoint death/discharge).

Illustration

The aim of the present section is to analyse the influence of a time-dependent covariate, namely ventilation, which is binary and reversible. We use the model of Figure 1 to analyse the data, where ventilation is considered as an intermediate event in an illness–death–model (Andersen et al., 1993, Chapter I.3.3).

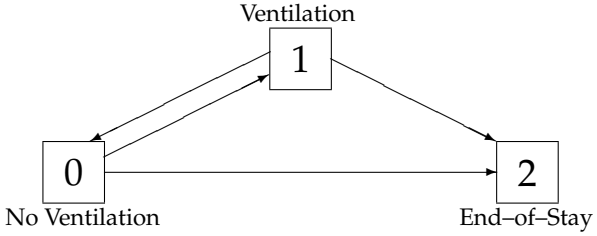


Figure 1: Model to assess the influence of ventilation.

The following call computes the estimated cumulative hazards along with the estimated variance

```
> data(sir.cont)
> na.cont <- mvna(sir.cont,c("0","1","2"),
+                   tra,"cens")
```

`tra` being a quadratic matrix of logical specifying the possible transitions:

```
> tra
      0     1     2
0 FALSE  TRUE  TRUE
1  TRUE FALSE  TRUE
2 FALSE FALSE FALSE
```

To assess the influence of the time–dependent covariate ventilation on the hazard of end–of–stay, we are specifically interested in the transitions from state "No ventilation" to "End–of–stay" and from state "Ventilation" to "End–of–stay". We specify to plot only these cumulative hazard estimates with the `tr.choice` option of the `xyplot.mvna` function.

```
> xyplot(na.cont,tr.choice=c("0 2","1 2"),
+ aspect=1,strip=strip.custom(bg="white",
+ factor.levels=
+    c("No ventilation -- Discharge/Death",
+      "Ventilation -- Discharge/Death"),
+ par.strip.text=list(cex=0.9)),
+ scales=list(alternating=1),xlab="Days",
+ ylab="Nelson-Aalen estimates")
```

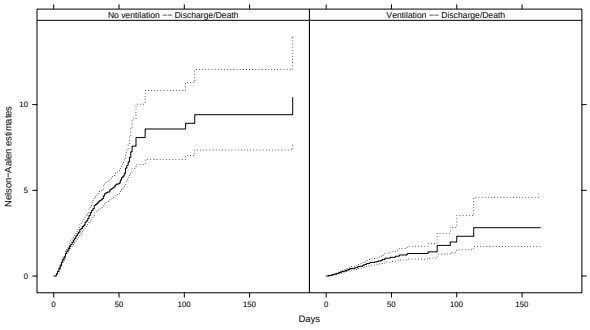


Figure 2: Nelson–Aalen estimates for the transitions "No ventilation" to "End–of–Stay", and "Ventilation" to "End–of–Stay".

We conclude from Figure 2 that ventilation prolongs the length of ICU stay since it reduces the hazard of end–of–stay.

Fig. 2 is well suited to illustrate a Cox analysis of ventilation as a time–dependent covariate (Therneau and Grambsch, 2001). Taking no ventilation as the baseline, we find a hazard ratio of 0.159 (95% CI, 0.132–0.193), meaning that ventilation prolongs hospital stay. Note that, in general, probability plots are not available to illustrate the analysis.

Variance estimation

`mvna` computes two variance estimators based on the optional variation process (Andersen et al., 1993, eq. (4.1.6)) and the predictable variation process (Andersen et al., 1993, eq. (4.1.7)) 'attached' to the Nelson-Aalen estimator, respectively. For standard survival data, Klein (1991) found that the (4.1.6) estimator tended to overestimate the true variance, but that the bias was of no practical importance for risk sets ≥ 5. He found that the (4.1.7) estimator tended to underestimate the true variance and had a smaller mean squared error. The two estimators appear to coincide from a practical point of view for risk sets ≥ 10. Small risk sets are a concern in multistate modelling because of multiple transient states. In a preliminary investigation, we found comparable results in a multistate model simulation. As in Klein (1991), we recommend the use of the (4.1.6) estimator, which was found to have a smaller absolute bias.

Summary

The **mvna** package provides a way to easily estimate and display the cumulative transition hazards from a time–inhomogeneous Markov multistate model. The estimator may remain valid under even more general assumptions; we refer to Andersen et al. (1993, Chapter III.1.3) and Glidden (2002) for mathematically detailed accounts. We hope that the **mvna** package will help to promote a computationally simple estimator that is extremely useful in illustrating and understanding complex event history processes, but that is underused in applications. We have illustrated the usefulness of the package for visualising the impact of a time–dependent covariate on survival. We also wish to mention that the package similarly illustrates standard Cox competing risks analyses in a straightforward way, whereas the usual cumulative incidence function plots don't; this issue is pursued in Beyersmann and Schumacher (2008). In closing, we also wish to mention the very useful R-package **muhaz** (Gentleman) for producing a smooth estimate of the survival hazard function, which also allows to estimate the univariate Nelson–Aalen estimator subject to right-censoring, but not to left-truncation.

Bibliography

P. K. Andersen, Ø. Borgan, R. D. Gill, and N. Keiding. *Statistical models based on counting processes*. Springer-Verlag, New-York, 1993.

J. Beyersmann and M. Schumacher. Time-dependent covariates in the proportional subdistribution hazards model for competing risks. *Biostatistics*, 2008.

J. Beyersmann, P. Gastmeier, H. Grundmann, S. Bärwolff, C. Geffers, M. Behnke, H. Rüden, and M. Schumacher. Use of Multistate Models to Assess Prolongation of Intensive Care Unit Stay Due to Nosocomial Infection. *Infection Control and Hospital Epidemiology*, 27:493–499, 2006a.

J. Beyersmann, T. Gerds, and M. Schumacher. Letter to the editor: comment on 'Illustrating the impact of a time-varying covariate with an extended Kaplan-Meier estimator' by Steven Snapinn, Qi Jiang, and Boris Iglewicz in the November 2005 issue of The American Statistician. *The American Statistician*, 60(30):295–296, 2006b.

E. J. Feuer, B. F. Hankey, J. J. Gaynor, M. N. Wesley, S. G. Baker, and J. S. Meyer. Graphical representation of survival curves associated with a binary non-reversible time dependent covariate. *Statistics in Medicine*, 11:455–474, 1992.

R. Gentleman. *muhaz: Hazard Function Estimation in Survival Analysis*. R package version 1.2.3.

D. Glidden. Robust inference for event probabilities with non-Markov data. *Biometrics*, 58:361–368, 2002.

J. D. Kalbfleisch and R. L. Prentice. *The Statistical Analysis of Failure Time Data*. John Wiley & Sons, 2002.

J. P. Klein. Small sample moments of some estimators of the variance of the Kaplan–Meier and Nelson–Aalen estimators. *Scandinavian Journal of Statistics*, 18:333–340, 1991.

T. Lumley. The survival package. *R News*, 4(1):26–28, June 2004. URL `http://CRAN.R-project.org/doc/Rnews/`.

D. Sarkar. Lattice. *R News*, 2(2):19–23, June 2002. URL `http://CRAN.R-project.org/doc/Rnews/`.

T. Therneau and P. Grambsch. *Modeling Survival Data: Extending the Cox Model*. Statistics for biology and health. Springer, New York, 2001.

Arthur Allignol
Jan Beyersmann
Martin Schumacher

Freiburg Center for Data Analysis and Modelling, University of Freiburg, Germany.
Institute of Medical Biometry and Medical Informatics, University Medical Centre Freiburg, Germany.

`arthur.allignol@fdm.uni-freiburg.de`

Programmers' Niche: The Y of R

Vince Carey

Introduction

Recursion is a general method of function definition involving self-reference. In R, it is easy to create recursive functions. The following R function computes the sum of numbers from 1 to n inclusive:

```
>   s <- function(n) {
+     if (n == 1) return(1)
+     return(s(n-1)+n)
+   }
```

Illustration:

```
>   s(5)

[1] 15
```

We can avoid the fragility of reliance on the function name defined as a global variable, by invoking the function through `Recall` instead of using self-reference.

It is easy to accept the recursion/`Recall` facility as a given feature of computer programming languages. In this article we sketch some of the ideas that allow us to implement recursive computation without named self-reference. The primary inspiration for these notes is the article "The Why of Y" by Richard P. Gabriel (Gabriel, 1988).

Notation. Throughout we interpret $=$ and `=` as the symbol for mathematical identity, and use `<-` to denote programmatic assignment, with the exception of notation for indices in algebraic summations (e.g., $\sum_{i=1}^{n}$), to which we give the usual interpretation.

The punch line

We can define a function known as the applicative-order fixed point operator for functionals as

```
> Y <- function(f) {
+     g <- function(h) function(x) f(h(h))(x)
+     g(g)
+ }
```

The following function can be used to compute cumulative sums in conjunction with `Y`:

```
>   csum <- function(f) function(n) {
+     if (n < 2) return(1);
+     return(n+f(n-1))
+   }
```

This call

```
>   Y(csum)(10)

[1] 55
```

computes $\sum_{k=1}^{10} k$ without iteration and without explicit self-reference.

Note that `csum` is not recursive. It is a function that accepts a function `f` and returns a function of one argument. Also note that *if* the argument passed to `csum` is the "true" cumulative sum function, which we'll denote K, then `csum(K)(n)` will be equal to `K(n)`. In fact, since we "know" that the function `s` defined above correctly implements cumulative summation, we can supply it as an argument to `csum`:

```
>    csum(s)(100)

[1] 5050
```

and we can see that

```
>    csum(s)(100) == s(100)

[1] TRUE
```

We say that a function f is a fixed point of a functional F if $F(f)(x) = f(x)$ for all relevant x. Thus, in the calculation above example, we exhibit an instance of the fact that `s` is a fixed point of `csum`. This is an illustration of the more general fact that K (the true cumulative summation function) is a fixed point of `csum` as defined in R.

Now we come to the crux of the matter. `Y` computes a fixed point of `csum`. For example:

```
> csum(Y(csum))(10) == Y(csum)(10)

[1] TRUE
```

We will show a little later that `Y` is the applicative-order fixed point operator for functionals `F`, meaning that `Y(F)(x)` $=$ `F(Y(F))(x)` for all suitable arguments `x`. This, in conjunction with a uniqueness theorem for "least defined fixed points for functionals", allows us to argue that K (a fixed point of `csum`) and `Y(csum)` perform equivalent computations. Since we can't really implement K (it is an abstract mathematical object that maps, in some unspecified way, the number n to $\sum_{k=1}^{n} k$) it is very useful to know that we can (and did) implement an equivalent function in R.

Peeking under the hood

One of the nice things about R is that we can interactively explore the software we are using, typically by mentioning the functions we use to the interpreter.

```
> s

function(n) {
    if (n == 1) return(1)
    return(s(n-1)+n)
  }
```

We have argued that `Y(csum)` is equivalent to *K*, so it seems reasonable to define

```
> K <- Y(csum)
> K(100)

[1] 5050
```

but when we mention this to the interpreter, we get

```
> K

function (x)
f(h(h))(x)
<environment: 0x1494d48>
```

which is not very illuminating. We can of course drill down:

```
> ls(environment(K))

[1] "h"

> H <- get("h", environment(K))
> H

function (h)
function(x) f(h(h))(x)
<environment: 0x1494e0c>

> get("f", environment(H))

function(f) function(n) {
    if (n < 2) return(1);
    return(n+f(n-1))
  }

> get("g", environment(H))

function (h)
function(x) f(h(h))(x)
<environment: 0x1494e0c>
```

The lexical scoping of R (Gentleman and Ihaka, 2000) has given us a number of closures (functions accompanied by variable bindings in environments) that are used to carry out the concrete computation specified by `K`. `csum` is bound to `f`, and the inner function of `Y` is bound to both `h` and `g`.

Self-reference via self-application

Before we work through the definition of `Y`, we briefly restate the punch line: `Y` transforms a functional having *K* as fixed point into a function that implements a recursive function that is equivalent to *K*. We want to understand how this occurs.

Define

```
>  ss <- function(f)function(n) {
+    if (n==1) return(1)
+    return(n+f(f)(n-1))
+ }
>  s(100)  # s() defined above
[1] 5050

>  ss(ss)(100)

[1] 5050
```

`s` is intuitively a recursive function with behavior equivalent to *K*, but relies on the interpretation of `s` in the global namespace at time of execution.

`ss(ss)` computes the same function, but avoids use of global variables. We have obtained a form of self-reference through self-application. This could serve as a reasonable implementation of *K*, but it lacks the attractive property possessed by `csum` that `csum(K)(x)` = `K(x)`.

We want to be able to establish functional self-reference like that possessed by `ss`, but without requiring the seemingly artificial self-application that is the essence of `ss`.

Note that `csum` can be self-applied, but only under certain conditions:

```
> csum(csum(csum(csum(88))))(4)

[1] 10
```

If the outer argument exceeded the number of self-applications, such a call would fail. Curiously, the argument to the innermost call is ignored.

Y

We can start to understand the function of `Y` by expanding the definition with argument `csum`. The first line of the body of `Y` becomes

```
g <- function(h) function(x) csum(h(h))(x)
```

The second line can be rewritten:

```
g(g) = function(x) csum(g(g))(x)
```

because by evaluating `g` on argument `g` we get to remove the first `function` instance and substitute `g` for `h`.

If we view the last "equation" syntactically, and pretend that `g(g)` is a name, we see that `Y(csum)` has created something that looks a lot like an instance of recursion via named self-reference. Let us continue with this impression with some R:

```
> cow <- function(x) csum(cow)(x)
> cow(100)

[1] 5050
```

`Y` has arranged bindings for constituents of `g` in its definition so that the desired recursion occurs without reference to global variables.

We can now make good on our promise to show that `Y` satisfies the fixed point condition `Y(F)(x)` = `F(Y(F))(x)`. Here we use an R-like notation to express formal relationships between constructs described above. The functional `F` is of the form

```
F = function(f) function(x) m(f,x)
```

where `m` is any R function of two parameters, of which the first may be a function. For any R function `r`,

```
F(r) = function(x) m(r,x)
```

Now the expression `Y(F)` engenders

```
g = function(h) function(x)F(h(h))(x)
```

and returns the value `g(g)`. This particular expression binds `g` to `h`, so that the value of `Y(F)` is in fact

```
function(x) F(g(g))(x)
```

Because `g(g)` is the value of `Y(F)` we can substitute in the above and find

```
Y(F) = function(x)Y(F)(x)
     = function(x) F(Y(F))(x)
```

as claimed.

Exercises

1. There is nothing special about recursions having the simple form of `csum` discussed above. Interpret the following mystery function and show that it can be modified to work recursively with `Y`.

```
> myst = function(x) {
+     if (length(x) == 0)
+         return(x)
+     if (x[1] %in% x[-1])
+         return(myst(x[-1]))
+     return(c(x[1], myst(x[-1])))
+ }
```

2. Extend `Y` to handle mutual recursions such as Hofstader's male-female recursion:

$$F(0) = 1, M(0) = 0,$$

$$F(n) = n - M(F(n-1)),$$

$$M(n) = n - F(M(n-1)).$$

3. Improve R's debugging facility so that debugging `K` does not lead to an immediate error.

Bibliography

R. Gabriel. The why of Y. *ACM SIGPLAN Lisp Pointers*, 2:15–25, 1988.

R. Gentleman and R. Ihaka. Lexical scope and statistical computing. *JCGS*, 9:491–508, 2000.

Changes in R Version 2.8.0

by the R Core Team

Significant user-visible changes

- `var()`, `cov()`, `cor()`, `sd()` etc. now by default (when 'use' is not specified) return NA in many cases where they signalled an error before.

New features

- `abbreviate()` gains an optional argument 'strict' allowing cheap and fast strict abbrevation.
- The "lm" methods of `add1()`, `anova()` and `drop1()` warn if they are mis-used on an essentially exact fit.
- `as.array()` is now generic, gains a '...' argument.
- New function `as.hexmode()` for converting integers in hex format. `format.hexmode()` and `as.character.hexmode()` gain an 'upper.case' argument.
- `bitmap()` and `dev2bitmap()` gain support for anti-aliasing. The default type has been changed to 'png16m', which supports anti-aliasing.
- `Box.test()` gains a 'fitdf' argument to adjust the degrees of freedom if applied to residuals.
- `browseURL()` has a new argument 'encodeIfNeeded' to use `URLencode()` in cases where it seems likely that would be helpful. (Unfortunately, those are hard to guess.)
- `by()` gains a 'simplify' argument, passed to `tapply()`.
- `capabilities()` gains a new argument "tiff" to report if `tiff()` is operational.
- `chol2inv()` now treats `<numeric(1)>` as a [1 x 1]-matrix.
- `cov()` and `cor()` have the option 'use = "everything"' as default, and so does `var()` with its default 'na.rm = FALSE'. This returns NA instead of signalling an error for NA observations. Another new option is 'use = "na.or.complete"' which is the default for `var(*, na.rm=FALSE)`. `var(double(0), na.rm = L)` now returns NA instead of signalling an error, for both L = TRUE or FALSE, as one consequence of these changes.
- `data.matrix()` now tries harder to convert non-numeric columns, via `as.numeric()` or `as(, "numeric")`.
- `dev.interactive()` is able to recognize the standard screen devices if `getOption("device")` is a function (as well as by name).
- `dev.new()` gains a '...' argument which can be used to pass named arguments which will be used if appropriate to the device selected.
- `dimnames(x) <-` value extends 'value' if it a list and too short, and 'x' is an array. This allows constructions such as `dimnames(x)[[1]] <- 1:3` to work whether or not 'x' already has dimnames.
- `format()`, `formatC()` and `prettyNum()` gain a new argument 'drop0trailing' which can be used to suppress trailing "0"s.
- `format()` now works for environments; also `print(env)` and `str(env)` share the same code for environments.
- It is now possible to create and open a text-mode `gzfile()` connection by explicitly using e.g. open="rt".
- New `help.request()` function for compiling an e-mail to R-help according to "the rules". It is built on the new utility, `create.post()` on which also `bug.report()` is based now; both thanks to a contribution by Heather Turner.
- `help.search()` now assumes that non-ASCII items are in latin1 if that makes sense (all known examples on CRAN are).
- `HoltWinters()` and `decompose()` use a (statistically) more efficient computation for seasonal fits (they used to waste one period).
- `intToUtf8()` and `intToBits()` now accept numeric vectors, truncating them to integers.
- `is.unsorted()` gains an argument 'strictly'. It now works for classed objects with a >= or > method (as incorrectly documented earlier).
- `library()` no longer warns about masking objects that are `identical(.,.)` to those they mask.
- `lockBinding()`, `unlockBinding()`, `lockEnvironment()` and `makeActiveBinding()` now all return invisibly (they always return NULL).
- `mood.test()` now behaves better in the presence of ties.

- `na.action()` now works on fits of classes "lm", "glm", ...
- `optim(.., method="SANN", .., trace=TRUE)` is now customizable via the 'REPORT' control argument, thanks to code proposals by Thomas Petzoldt.
- The 'factory-fresh' defaults for `options("device")` have been changed to refer to the devices as functions in the grDevices namespace and not as names. This makes it more likely that the incorrect (since R 2.5.0) assumption in packages that `get(getOption("device"))()` will work will catch users of those packages.
- pch=16 now has no border (for consistency with 15, 17, 18) and hence is now different from pch=19.
- `pdf()` has new arguments 'useDingbats' (set this to FALSE for use with broken viewers) and 'colormodel'. It now only references the ZapfDingbats font if it is used (for small opaque circles).

  The default PDF version is now 1.4, since viewers that do not accept that are now rare.

  Different viewers were rendering consecutive `text()` calls on a `pdf()` device in different ways where translucency was involved. The PDF generated has been changed to force each call to be rendered separately (which is the way xpdf or ghostscript was rendering, but Acrobat was forming a transparency group), which is consistent with other graphics devices supporting semi-transparency.
- `plot.dendrogram()` has new arguments (xlim, ylim) which allows zooming into a hiearchical clustering dendrogram.
- `plot.histogram()` gains an 'ann' argument. (Wish from Ben Bolker.)
- `plot(<lm_obj>)` now warns when it omits points with leverage one from a plot.
- Plotmath now recognizes 'aleph' and 'nabla' (the Adobe Symbol 'gradient' glyph) as symbol names.
- `polyroot()` no longer has a maximum degree.
- The alpha/alphamax argument of the 'nls' and 'mle' `profile()` methods is used to compute confidence limits for univariate t-statistics rather than a confidence region for all the parameters (and not just those being profiled).
- `quantile.default()` allows 'probs' to stray just beyond [0, 1], to allow for computed values.
- New functions `rawConnection()` and `rawConnectionValue()` allow raw vectors to be treated as connections.
- `read.dcf()` now consistently gives an error for malformed DCF.
- `read.fwf()` no longer passes its default for 'as.is' to `read.table()`: this allows the latter's default to be used.
- `readBin()`, `writeBin()`, `readChar()` and `writeChar()` now open a connection which was not already open in an appropriate binary mode rather than the default mode.

  `readLines()`, `cat()` and `sink()` now open a connection which was not already open in an appropriate text mode rather than the default mode.
- `readCitationFile()` (and hence citation) now reads a package's CITATION file in the package's declared encoding (if there is one).
- The behaviour of `readLines()` for incomplete final lines on binary-mode connections has been changed to be like blocking rather than non-blocking text-mode connections.
- A new `reorder.character()` method has been added. This allows use of `'reorder(x, ...)'` as a shorthand for `'reorder(factor(x), ...)'` when 'x' is a character vector.
- `round()` now computes in long doubles where possible so the results are more likely to be correct to representation error.
- `rug()` now uses `axis()`'s new arguments from 2.7.2, hence no longer draws an axis line.
- `save()` (optionally, but by default) checks for the existence of objects before opening the file/connections (wish of PR#12543).
- `segments()`, `arrows()` and `rect()` allow zero-length coordinates. (Wish of PR#11192.)
- `set.seed(kind=NULL)` now takes 'kind' from a saved seed if the workspace has been restored or .Random.seed has been set in some other way. Previously it took the 'currently used' value, which was "default" unless random numbers had been used in the current session. Similarly for the values reported by `RNGkind()`. (Related to PR#12567.)

  `set.seed()` gains a 'normal.kind' argument.
- `setEPS()` and `setPS()` gain '...' to allow other arguments to be passed to `ps.options()`, including overriding 'width' and 'height'.

- `setTimeLimit()` function to set limits on the CPU and/or elapsed time for each top-level computation, and `setSessionLimit()` to set limits for the rest of the session.
- `splinefun()` has a new method = "monoH.FC" for monotone Hermite spline interpolation.
- `sprintf()` optionally supports the %a/%A notation of C99 (if the platform does, including under Windows).
- `str()`'s default method gains a 'formatNum' function argument which is used for formatting numeric vectors. Note that this is very slightly not backward compatible, and that its default may change before release.
- The `summary()` method for class "ecdf" now uses a `print()` method rather than printing directly.
- `summary.manova()` uses a stabler computation of the test statistics, and gains a 'tol' argument to allow highly correlated responses to be explored (with probable loss of accuracy). Similar changes have been made to `anova.mlm()` and `anova.mlmlist()`.
- `Sweave()` now writes concordance information inside a `\Sconcordance` LaTeXmacro, which allows it to be inserted into PDF output.
- `system.time()` now uses lazy evaluation rather than eval/substitute, which results in more natural scoping. (PR#11169.)
- In `table()`, 'exclude=NULL' now does something also for factor arguments. A new 'useNA' argument allows you to control whether to add NA levels unconditionally or only when present in data. A new convenience function `addNA()` gives similar functionality by adding NA levels to individual factors.
- `unlink()` tries the literal pattern if it does not match with wildcards interpreted – this helps with e.g. `unlink("a[b")` which previously needed to be `unlink("a\\[b")`.
- `update.packages()` gains an argument 'oldPkgs', where `new.packages()` and `old.packages()` get 'instPkgs'. These allow to consider only subsets of packages instead of all installed ones.
- `which(b)` is somewhat faster now, notably for named vectors, thanks to a suggestion by Henrik Bengtsson.
- New generic function `xtfrm()` as an auxiliary helper for `sort()`, `order()` and `rank()`. This should return a numeric vector that sorts in the same way as its input. The default method supports any class with ==, > and `is.na()` methods but specific methods can be much faster.

  As a side-effect, `rank()` will now work better on classed objects, although possibly rather slowly.
- `X11()` and `capabilities("X11")` now catch some X11 I/O errors that previously terminated R. These were rare and have only be seen with a misconfigured X11 setup on some versions of X11.
- The handling of nuls in character strings has been changed – they are no longer allowed, and attempting to create such a string now gives a truncation warning (unless `options("warnEscapes")` is FALSE).
- The user environment and profile files can now be specified via environment variables 'R_ENVIRON_USER' and 'R_PROFILE_USER', respectively.
- ?pkg::topic and ?pkg:::topic now find help on 'topic' from package 'pkg' (and not help on :: or :::).
- ??topic now does `help.search("topic")`; variations such as ??pkg::topic or field??topic are also supported.
- There is support for using ICU (International Components for Unicode) for collation, enabled by configure option –with-ICU on a Unix-alike and by a setting in MkRules on Windows. Function `icuSetCollate()` allows the collation rules (including the locale) to be tuned. [Experimental.]
- If S4 method dispatch is on and S4 objects are found as attributes, `show()` rather than `print()` is used to print the S4 attributes.
- Starting package tcltk without access to Tk (e.g. no available display) is now a warning rather than an error, as Tcl will still be usable. (On most platforms it was possible to inhibit Tk by not having DISPLAY set, but not on Windows nor Mac OS X builds with –with-aqua.)
- Using $ on a non-subsettable object (such as a function) is now an error (rather than returning NULL).
- Hexadecimal numerical constants (such as 0xab.cdp+12) may now contain a decimal point.
- PCRE has been updated to version 7.8 (mainly bug fixes).
- `plot.ecdf()` now defaults to pch=19 so as to better convey the left-closed line segments.

New features in package 'methods'

- S3 classes that are registered by a call to `setOldClass()` now have the S3 class as a special slot, and therefore so do any S4 classes that contain them. This mechanism is used to support S4 classes that extend S3 classes, to the extent possible. See ?Classes, ?setOldClass, and ?S3Class.

  The treatment of special pseudo-classes "matrix", "array", and "ts" as S4 classes has also been modified to be more consistent and, within limitations imposed by special treatment of these objects in the base code, to allow other classes to contain them. See class?ts.

  A general feature added to implement "ts" and also "data.frame" as S4 classes is that an S4 class definition can be supplied to `setOldClass()` when the S3 class has known attributes of known class.

  `setOldClass()` now saves all the S3 inheritance, allowing the calls to be built up in stages, rather than including all the S3 classes in each call. Also allows `as(x,"S3")` to generate valid S3 inheritance from the stored definition. See ?S3.

- S4 methods may now be defined corresponding to "...", by creating a generic function that has "..." as its signature. A method will be selected and called if all the arguments matching "..." are from this class or a subclass. See ?dotsMethods.

- New functions `S3Part()` and `S3Class()` provide access to the corresponding S3 object and class for S4 classes that extend either an S3 class or a basic R object type.

- `show(<class definition>)` now also shows the class name.

Installation

- If sub-architectures are used, a copy of Rscript is installed in $R_HOME/bin/exec$R_ARCH (since that in $R_HOME/bin and /usr/bin might be overwritten in a subsequent installation).

Package installation

- LazyLoad: yes is now the default, so packages wanting to avoid lazy loading must set 'LazyLoad: no' (or an equivalent value) in the DESCRIPTION file.

- R CMD INSTALL will now fail if it finds a non-executable 'configure' script in the package – this usually indicates a file system with insufficient permissions. If a non-executable 'cleanup' script is found and either –clean or –preclean is used, a warning is given.

Deprecated & defunct

- Use in packages of the graphics headers Rdevices.h and Rgraphics.h is defunct: they are no longer installed.

- `options("par.ask.default")` is defunct in favour of "device.ask.default".

- The 'device-independent' family "symbol" is defunct: use font=5 (base) or fontface=5 (grid) instead.

- `gammaCody()` is defunct.

- `par("gamma")` is defunct.

- 'methods' package functions `getAccess()`, `getAllMethods()`, `getClassName()`, `getClassPackage()`, `getExtends()`, `getProperties()`, `getPrototype()`, `getSubclasses()`, `getVirtual()`, `mlistMetaName()`, `removeMethodsObject()` and `seemsS4Object()` are defunct.

- Use of a non-integer .Random.seed is now an error. (R itself has never generated such values, but user code has, and R >= 2.6.0 has given a warning.)

- `methods::allGenerics()` is deprecated.

- In package installation, SaveImage: yes is now ignored, and any use of the field will give a warning.

- `unserialize()` no longer accepts character strings as input.

- The C macro 'allocString' has been removed – use 'mkChar' and variants.

- Use of `allocVector(CHARSXP ...)` is deprecated and gives a warning.

Utilities

- The default for 'stylepath' in Sweave's (default) RweaveLatex driver is now FALSE rather than TRUE if SWEAVE_STYLEPATH_DEFAULT is unset : see ?RweaveLatex. To support this, tools::texi2dvi adds the R 'texmf' directory to the input search path.

- R CMD Rd2dvi now previews PDF output (as was documented) if R_PDFVIEWER is set (as it will normally be on a Unix-alike but not on Windows, where the file association is used by default).
- R CMD check checks for binary executable files (which should not appear in a source package), using a suitable 'file' if available, else by name.
- R CMD check now also uses codetools' checks on the body of S4 methods.

C-level facilities

- R_ReadConsole will now be called with a buffer size of 4096 bytes (rather than 1024): maintainers of alternative front-ends should check that they do not have a smaller limit.
- Graphics structure NewDevDesc has been renamed to DevDesc. For now there is a compatibility define in GraphicsDevice.h, but it will be removed in R 2.9.0.
- PROTECT and UNPROTECT macros now work even with R_NO_REMAP.

Bug fixes

- @ now gives an error (and not just a warning) if it is being applied to a non-S4 object.
- R CMD appends (not prepends) R's texmf path to TEXINPUTS.
- Objects generated by `new()` from S4 classes should now all satisfy `isS4(object)`. Previously, prototypes not of object type S4 would not be S4 objects. `new()` applied to basic, non-S4 classes still will (and should) return non-S4 objects.
- Functions writing to connections such as `writeLines()`, `writeBin()`, `writeChar()`, `save()`, `dput()` and `dump()` now check more carefully that the connections are opened for writing, including connections that they open themselves.

  Similarly functions which read such as `readLines()`, `scan()`, `dcf()` and `parse()` check connections for being open for reading.
- Equality comparison of factors with <NA> levels now works correctly again.
- Repainting of open X11 `View()` windows is now done whilst an X11 dataentry window is in use.
- Indexing of data frames with NA column names and a numeric or logical column index works again even if columns with NA names are selected.
- `on.exit()` has been fixed to use lexical scope in determining where to evaluate the exit action when the on.exit expression appears in a function argument.
- `rank()` now consistently returns a double result for ties.method = "average" and an integer result otherwise. Previously the storage mode depended on 'na.last' and if any NAs were present.
- The "lm" methods of `add1()`, and `drop1()` now also work on a model fit with na.action = na.exclude.
- `median(c(x = NA_real_))` no longer has spurious `names()`.
- `isoreg(x, y)` now returns the correct result also when x has ties, in all cases.
- What `na.action()` does is now correctly documented.
- `source()` with echo=TRUE now behaves like ordinary automatic printing, by using `methods::show()` for S4 objects.
- Several bugs fixed in '?' with topics: it previously died trying to construct some error messages; for S4 methods, class "ANY" should be used for omitted arguments and default methods.
- `trace()` should create missing traceable classes in the global environment, not in `baseenv()` where other classes will not be found.
- Class inheritance using explicit coerce= methods via `setIs()` failed to coerce the argument in method dispatch. With this fixed, a mechanism was needed to prohibit such inheritance when it would break the generic function (e.g., initialize). See ?setIs and ?setGeneric.
- `RSiteSearch()` encodes its query (it seems this is occasionally needed on some platforms, but encoding other fields is harmful).
- 'incomparables' in `match()` was looking up indices in the wrong table.
- `write.dcf()` did not escape "." according to Debian policy (PR#12816).
- `col2rgb()` sometimes opened a graphics device unnecessarily, and `col2rgb(NA)` did not return a transparent color, as documented.

- `pdf(family="Japan")` [and other CIDfonts] no longer seg.faults when writing "western" text strings.
- `as.list()` applied to an environment now forces promises and returns values.
- Promises capturing calls to `sys.parent()` and friends did not work properly when evaluated via method dispatch for internal S3 generics.
- The default pkgType option for non-CRAN builds of R on Mac OS X is now correctly "source" as documented.
- The default R_LIBS_USER path in AQUA builds now matches the Mac-specifc path used by the Mac GUI: /Library/R/x.y/library
- `splinefun()` with natural splines incorrectly evaluated derivatives to the left of the first knot. (PR#13132, fix thanks to Berwin Turlach.)
- `anova(glm(..., y=FALSE))` now works. (PR#13098.)
- `cut.Date(x, "weeks")` could fail if x has only one unique value which fell on a week boundary. (PR#13159.)

Changes on CRAN

by Kurt Hornik

New contributed packages

AER Functions, data sets, examples, demos, and vignettes for the book "Applied Econometrics with R" by C. Kleiber and A. Zeileis (2008, Springer-Verlag, New York, ISBN 978-0-387-77316-2). By Achim Zeileis and Christian Kleiber.

ALS Multivariate curve resolution alternating least squares (MCR-ALS). Alternating least squares is often used to resolve components contributing to data with a bilinear structure; the basic technique may be extended to alternating constrained least squares. Commonly applied constraints include unimodality, non-negativity, and normalization of components. Several data matrices may be decomposed simultaneously by assuming that one of the two matrices in the bilinear decomposition is shared between datasets. By Katharine M. Mullen.

AdMit Performs fitting of an adaptive mixture of Student t distributions to a target density through its kernel function. The mixture approximation can then be used as the importance density in importance sampling or as the candidate density in the Metropolis-Hastings algorithm to obtain quantities of interest for the target density itself. By David Ardia, Lennart F. Hoogerheide and Herman K. van Dijk.

BCE Estimation of taxonomic compositions from biomarker data, using a Bayesian approach. By Karel Van den Meersche and Karline Soetaert.

BLCOP An implementation of the Black-Litterman Model and Atilio Meucci's copula opinion pooling framework. By Francisco Gochez.

BaM Functions and datasets for "Bayesian Methods: A Social and Behavioral Sciences Approach" by J. Gill (Second Edition, 2007, CRC Press). By Jeff Gill.

BiplotGUI Provides a GUI with which users can construct and interact with biplots. By Anthony la Grange.

CADStat Provides a GUI to several statistical methods useful for causal assessment. Methods include scatterplots, boxplots, linear regression, generalized linear regression, quantile regression, conditional probability calculations, and regression trees. By Lester Yuan, Tom Stockton, Doug Bronson, Pasha Minallah and Mark Fitzgerald.

CHsharp Functions that cluster 3-dimensional data into their local modes. Based on a convergent form of Choi and Hall's (1999) data sharpening method. By Douglas G. Woolford.

CTT Classical Test Theory Functions. By John T. Willse and Zhan Shu.

ConvergenceConcepts Provides a way to investigate various modes of convergence of random variables. By P. Lafaye de Micheaux and B. Liquet.

CvM2SL2Test Cramer-von Mises Two Sample Tests, featuring functionality to compute the exact p-value(s) for given Cramer-von Mises two-sample test score(s) under the assumption that the populations under comparison have the same probability distribution. By Yuanhui Xiao.

DiagnosisMed Analysis of data from diagnostic test accuracy evaluating health conditions. Designed for use by health professionals. Can estimate sample size for common situations in diagnostic test accuracy, estimate sensitivity and specificity from categorical and continuous test results including some evaluations of indeterminate results, or compare different analysis strategies into measures commonly used by health professionals. By Pedro Brasil.

EVER Estimation of Variance by Efficient Replication, providing delete-a-group jackknife replication. Gives estimates, standard errors and confidence intervals for: totals, means, absolute and relative frequency distributions, contingency tables, ratios, quantiles and regression coefficients, as well as for for user-defined estimators (even non-analytic). By Diego Zardetto.

FITSio Utilities to read and write files in the FITS (Flexible Image Transport System) format, a standard format in astronomy. By Andrew Harris.

FTICRMS Programs for Analyzing Fourier Transform-Ion Cyclotron Resonance Mass Spectrometry Data. By Don Barkauskas.

GExMap Analysis of genomic distribution of genes lists produced by transcriptomic studies. By N. Cagnard.

HAPim Provides a set of functions whose aim is to propose 4 methods of QTL detection: HAPimLD (an interval-mapping method designed for unrelated individuals with no family information that makes use of linkage disequilibrium), HAPimLDL (an interval-mapping

method for design of half-sib families, combining linkage analysis and linkage disequilibrium), HaploMax (based on an analysis of variance with a dose haplotype effect), and HaploMaxHS (based on an analysis of variance with a sire effect and a dose haplotype effectin half-sib family design). By S. Dejean, N. Oumouhou, D. Estivals and B. Mangin.

LDtests Exact tests for Linkage Disequilibrium (LD) and Hardy-Weinberg Equilibrium (HWE). By Alex Lewin.

LIStest Compute the *p*-value for the Longest Increasing Subsequence Independence Test (for continuous random variables). By Jesus Garcia and Veronica Andrea Gonzalez Lopez.

MAclinical Class prediction based on microarray data and clinical parameters. Prediction is performed using a two-step method combining (pre-validated) PLS dimension reduction and random forests. By Anne-Laure Boulesteix.

MCAPS Weather and air pollution data, risk estimates, and other information from the Medicare Air Pollution Study (MCAPS) of 204 U.S. counties, 1999–2002. By Roger D. Peng.

MDD Calculates Minimum Detectable Difference (MDD) for several continuous and binary endpoints. Also contains programs to compare the MDD to the clinically significant difference for most of the same tests. By Don Barkauskas.

MIfuns Pharmacometric tools for common data preparation tasks, stratified bootstrap resampling of data sets, NONMEM control stream creation/editing, NONMEM model execution, creation of standard and user-defined diagnostic plots, execution and summary of bootstrap and predictive check results, implementation of simulations from posterior parameter distributions, reporting of output tables and creation of a detailed analysis log. By Bill Knebel.

MKmisc Miscellaneous Functions from M. Kohl. By Matthias Kohl.

MSVAR Estimation of a 2 state Markov Switching VAR. By James Eustace.

Metabonomic Graphical user interface for metabonomic analysis (baseline, normalization, peak detection, PCA, PLS, nearest neigbor, neural network). By Jose L. Izquierdo.

NetIndices Estimates network indices, including trophic structure of foodwebs. Indices include ascendency network indices, direct and indirect dependencies, effective measures, environ network indices, general network indices, pathway analysis, network uncertainty indices and constraint efficiencies and the trophic level and omnivory indices of food webs. By Karline Soetaert and Julius Kipkeygon Kones.

OPE Fit an outer-product emulator to the multivariate evaluations of a computer model. By Jonathan Rougier.

PBSddesolve Solver for Delay Differential Equations via interfacing numerical routines written by Simon N. Wood, with contributions by Benjamin J. Cairns. Replaces package **ddesolve**. By Alex Couture-Beil, Jon Schnute and Rowan Haigh.

PSM Estimation of linear and non-linear mixed-effects models using stochastic differential equations. Also provides functions for finding smoothed estimates of model states and for simulation. Allows for any multivariate non-linear time-variant model to be specified, and also handles multi-dimensional input, covariates, missing observations and specification of dosage regimen. By Stig Mortensen and Søren Klim.

PolynomF Univariate polynomial operations in R. By Bill Venables.

Pomic Calculations of Pattern Oriented Modeling Information Criterion (POMIC), a non-parsimonious based information criterion, to check the quality of simulations results of ABM/IBM or other non-linear rule-based models. The POMIC is based on the KL divergence and likelihood theory. By Cyril Piou.

PredictiveRegression Prediction intervals for three algorithms described in "On-line predictive linear regression" (Annals of Statistics, 2008). By Vladimir Vovk and Ilia Nouretdinov.

PtProcess Time dependent point process modeling, with an emphasis on earthquake modeling. By David Harte.

R2jags Running JAGS from R. By Yu-Sung Su and Masanao Yajima.

RItools Tools for randomization inference. By Jake Bowers, Mark Fredrickson and Ben Hansen.

RKEA An R interface to KEA (Version 5.0). By Ingo Feinerer.

RM2 Functions used in revenue management and pricing environments. By Tudor Bodea, Dev Koushik and Mark Ferguson.

RPostgreSQL Database interface and PostgreSQL driver for R. Complies with the database interface definition as implemented in package **DBI**. By Sameer Kumar Prayaga with mentor Dirk Eddelbuettel.

Ratings Functions to implement the methods described in "Improving the Presentation and Interpretation of Online Ratings Data with Model-based Figures" by Ho and Quinn (The American Statistician). By Kevin M. Quinn and Daniel E. Ho.

Read.isi Automated access to old World Fertility Survey data saved in fixed-width format based on ISI-formatted codebooks. By Rense Nieuwenhuis.

ResearchMethods Using GUIs to help teach statistics to non-statistics students. By Sam Stewart and Mohammed Abdolell.

RobAStBase Base S4 classes and functions for robust asymptotic statistics. By Matthias Kohl and Peter Ruckdeschel.

Rsge Functions for using R with the SGE cluster/grid queuing system. By Dan Bode.

SASPECT Significant AnalysiS of PEptide CounTs. A statistical method for significant analysis of comparative proteomics based on LC-MS/MS Experiments. By Pei Wang and Yan Liu.

SDDA Stepwsie Diagonal Discriminant Analysis — a fast algorithm for building multivariate classifiers. By CSIRO Bioinformatics, Glenn Stone.

SGP Calculate growth percentile and projections for students using large scale, longitudinal assessment data. These norm referenced growth values are presently used in some state testing and accountability systems. The functions use quantile regression techniques package to estimate the conditional density associated with each student's achievement history. Student Growth Projections (i.e., percentile growth trajectories) help to identify what it will take for students to reach future achievement targets. By Damian W. Betebenner, with contributions from Jonathan Weeks, Jinnie Choi, Xin Wei and Hi Shin Shim.

SMVar Structural Model for Variances in order to detect differentially expressed genes from gene expression data. By Guillemette Marot.

SNPMaP SNP Microarrays and Pooling in R. Pooling DNA on SNP microarrays is a cost-effective way to carry out genome-wide association studies for heritable disorders or traits. By Oliver SP Davis and Leo C Schalkwyk.

SNPMaP.cdm Annotation for SNP Microarrays and Pooling in R: provides cdm objects for the **SNPMaP** package. By Oliver SP Davis and Leo C Schalkwyk.

SQLiteMap Manage vector graphical maps using SQLite. By Norbert Solymosi.

STAR Spike Train Analysis with R: functions to analyze neuronal spike trains from a single neuron or from several neurons recorded simultaneously. By Christophe Pouzat.

SiZer 'SiZer: Significant Zero Crossings. Calculates and plots the SiZer map for scatterplot data. A SiZer map is a way of examining when the p-th derivative of a scatterplot-smoother is significantly negative, possibly zero or significantly positive across a range of smoothing bandwidths. By Derek Sonderegger.

SimpleTable Methods to conduct Bayesian inference and sensitivity analysis for causal effects from 2×2 and $2 \times 2 \times K$ tables when unmeasured confounding is present or suspected. By Kevin M. Quinn.

SpatialExtremes Several approaches to spatial extremes modeling. By Mathieu Ribatet.

StatMatch Perform statistical matching between two data sources and also for imputing missing values in data sets through hot deck methods. By Marcello D'Orazio.

StreamMetabolism Calculate single station metabolism from diurnal Oxygen curves. By Stephen A Sefick Jr.

TinnR Tinn-R GUI/editor resources for R. By Jose Claudio Faria, based on code by Philippe Grosjean.

TraMineR Sequences and trajectories mining for the field of social sciences, where sequences are sets of states or events describing life histories, for example family formation histories. Provides tools for translating sequences from one format to another, statistical functions for describing sequences and methods for computing distances between sequences using several metrics like optimal matching and some other metrics proposed by C. Elzinga. By Alexis Gabadinho, Matthias Studer, Nicolas S. Müller and Gilbert Ritschard.

VhayuR Vhayu R Interface. By The Brookhaven Group.

apsrtable Formats LaTeX tables from one or more model objects side-by-side with standard errors below, not unlike tables found in such journals as the American Political Science Review. By Michael Malecki.

audio Interfaces to audio devices (mainly sample-based) from R to allow recording and playback of audio. Built-in devices include Windows MM, Mac OS X AudioUnits and PortAudio (the last one is very experimental). By Simon Urbanek.

bark Bayesian Additive Regression Kernels. Implementation of BARK as described in Zhi Ouyang's 2008 Ph.D. thesis. By Zhi Ouyang, Merlise Clyde and Robert Wolpert.

bayesGARCH Bayesian estimation of the GARCH(1,1) model with Student's *t* innovations. By David Ardia.

bear An average bioequivalence (ABE) and bioavailability data analysis tool including sample size estimation, noncompartmental analysis (NCA) and ANOVA (lm) for a standard RT/TR 2-treatment, 2-sequence, 2-period, and balanced, cross-over design. By Hsin-ya Lee and Yungjin Lee.

betaper Evaluates and quantifies distance decay of similarity among biological inventories in the face of taxonomic uncertainty. By Luis Cayuela and Marcelino de la Cruz.

bigmemory Use C++ to create, store, access, and manipulate massive matrices. Under UNIX, it also supports use of shared memory. By Michael J. Kane and John W. Emerson.

bise Some easy-to-use functions for spatial analyses of (plant-) phenological data sets and satellite observations of vegetation. By Daniel Doktor.

bit A class for vectors of 1-bit booleans. With bit vectors you can store true binary booleans at the expense of 1 bit only; on a 32 bit architecture this means factor 32 less RAM and factor 32 more speed on boolean operations. By Jens Oehlschlägel.

bmd Benchmark dose analysis for continuous and quantal dose-response data. By Christian Ritz.

bpca Biplot (2d and 3d) of multivariate data based on principal components analysis and diagnostic tools of the quality of the reduction. By Jose Claudio Faria and Clarice Garcia Borges Demetrio.

ccgarch Estimation and simulation of Conditional Correlation GARCH (CC-GARCH) models. By Tomoaki Nakatani.

cem Coarsened Exact Matching. Implements the CEM algorithm (and many extensions) described in 'Matching for Causal Inference Without Balance Checking" by Stefano M. Iacus, Gary King, and Giuseppe Porro (`http://gking.harvard.edu/files/abs/cem-abs.shtml`). By Stefano Iacus, Gary King and Giuseppe Porro.

compare Functions for comparing vectors and data frames. By Paul Murrell.

compoisson Routines for density and moments of the Conway-Maxwell-Poisson distribution as well as functions for fitting the COM-Poisson model for over/under-dispersed count data. By Jeffrey Dunn.

convexHaz Functions to compute the nonparametric maximum likelihood estimator (MLE) and the nonparametric least squares estimator (LSE) of a convex hazard function, assuming that the data is i.i.d. By Hanna Jankowski, Ivy Wang, Hugh McCague and Jon A. Wellner.

copas Statistical methods to model and adjust for bias in meta-analysis. By James Carpenter and Guido Schwarzer.

coxphw Weighted estimation for Cox regression. R by Meinhard Ploner, FORTRAN by Georg Heinze.

curvetest Test if two curves defined by two data sets are equal, or if one curve is equal to zero. By Zhongfa Zhang and Jiayang Sun.

dataframes2xls Write data frames to '.xls' files. Supports multiple sheets and basic formatting. By Guido van Steen.

ddst Data driven smooth test. By Przemyslaw Biecek (R code) and Teresa Ledwina (support, descriptions).

deSolve General solvers for ordinary differential equations (ODE) and for differential algebraic equations (DAE). The functions provide an interface to the FORTRAN functions lsoda, lsodar, lsode, lsodes, dvode and daspk. The package also contains routines designed for solving uni- and multicomponent 1-D and 2-D reactive transport models. By Karline Soetaert, Thomas Petzoldt and R. Woodrow Setzer.

denstrip Graphical methods for compactly illustrating probability distributions, including density strips, density regions, sectioned density plots and varying width strips. By Christopher Jackson.

dfcrm Dose-finding by the continual reassessment method. Provides functions to run the CRM and TITE-CRM in phase I trials and calibration tools for trial planning purposes. By Ken Cheung.

diffractometry Residual-based baseline identification and peak decomposition for x-ray diffractograms as introduced in the corresponding paper by Davies et al. (2008, Annals of Applied Statistics). By P. L. Davies, U. Gather, M. Meise, D. Mergel and T. Mildenberger. Additional Code by T. Bernholt and T. Hofmeister.

dirichlet Dirichlet model of consumer buying behavior for marketing research. The Dirichlet (aka NBD-Dirichlet) model describes the purchase incidence and brand choice of consumer products. Provides model estimation and summaries of various theoretical quantities of interest to marketing researchers. Also provides functions for making tables that compare observed and theoretical statistics. By Feiming Chen.

distrMod Object orientated implementation of probability models based on packages **distr** and **distrEx**. By Matthias Kohl and Peter Ruckdeschel.

distrTeach Extensions of package **distr** for teaching stochastics/statistics in secondary school. By Peter Ruckdeschel, Matthias Kohl, Anja Hueller and Eleonara Feist.

divagis Tools for quality checks of georeferenced plant species accessions. By Reinhard Simon.

dti DTI Analysis. Diffusion Weighted Imaging is a Magnetic Resonance Imaging modality that measures diffusion of water in tissues like the human brain. The package contains unctions to process diffusion-weighted data in the context of the diffusion tensor model (DTI), including the calculation of anisotropy measures and, the implementation of the structural adaptive smoothing algorithm described in "Diffusion Tensor Imaging: Structural Adaptive Smoothing" by K. Tabelow, J. Polzehl, V. Spokoiny, and H.U. Voss (2008, Neuroimage 39(4), 1763-1773). By Karsten Tabelow and Joerg Polzehl.

dynamo Routines for estimation, simulation, regularization and prediction of univariate dynamic models including: ARMA, ARMA-GARCH, ACD, MEM. By Christian T. Brownlees.

ecolMod Figures, data sets and examples from the book "A practical guide to ecological modelling — using R as a simulation platform" by Karline Soetaert and Peter MJ Herman (2008, Springer). By Karline Soetaert and Peter MJ Herman.

empiricalBayes A bundle providing a simple solution to the extreme multiple testing problem by estimating local false discovery rates. Contains two packages, **localFDR** and **HighProbability**. Given a vector of p-values, the former estimates local false discovery rates and the latter determines which p-values are low enough that their alternative hypotheses can be considered highly probable. By Zahra Montazeri and David R. Bickel.

entropy Implements various estimators of entropy, such as the shrinkage estimator by Hausser and Strimmer, the maximum likelihood and the Millow-Madow estimator, various Bayesian estimators, and the Chao-Shen estimator. It also offers an R interface to the NSB estimator. Furthermore, it provides functions for estimating mutual information. By Jean Hauser and Korbinian Strimmer.

eqtl Analysis of experimental crosses to identify genes (called quantitative trait loci, QTLs) contributing to variation in quantitative traits, a complementary to Karl Broman's **qtl** package for genome-wide analysis. By Ahmid A. Khalili and Olivier Loudet.

etm Empirical Transition Matrix: matrix of transition probabilities for any time-inhomogeneous multistate model with finite state space. By Arthur Allignol.

expert Modeling without data using expert opinion. Expert opinion (or judgment) is a body of techniques to estimate the distribution of a random variable when data is scarce or unavailable. Opinions on the quantiles of the distribution are sought from experts in the field and aggregated into a final estimate. Supports aggregation by means of the Cooke, Mendel-Sheridan and predefined weights models. By Mathieu Pigeon, Michel Jacques and Vincent Goulet.

fbati Family-based gene by environment interaction tests, and joint gene, gene-environment interaction test. By Thomas Hoffmann.

fgui Function GUI: rapidly create a GUI for a function by automatically creating widgets for arguments of the function. By Thomas Hoffmann.

fossil Palaeoecological and palaeogeographical analysis tools. Includes functions for estimating species richness (Chao 1 and 2, ACE, ICE, Jacknife), shared species/beta diversity, species area curves and geographic distances and areas. By Matthew Vavrek.

frontier Maximum Likelihood Estimation of stochastic frontier production and cost functions. Two specifications are available: the error components specification with time-varying efficiencies (Battese and Coelli, 1992) and a model specification in which the firm effects are directly influenced by a number of variables (Battese and Coelli, 1995). By Tim Coelli and Arne Henningsen.

fts R interface to tslib (a time series library in C++). By Whit Armstrong.

fuzzyOP Fuzzy numbers and the main mathematical operations for these. By Aklan Semagul, Altindas Emine, Macit Rabiye, Umar Senay and Unal Hatice.

gWidgetsWWW Toolkit implementation of **gWidgets** for www. By John Verzani.

gbs Utilities for analyzing censored and uncensored data from generalized Birnbaum-Saunders distributions. By Michelli Barros, Victor Leiva and Gilberto A. Paula.

gene2pathway Prediction of KEGG pathway membership for individual genes based on InterPro domain signatures. By Holger Froehlich, with contributions by Tim Beissbarth.

geonames Interface to `www.geonames.org` web service. By Barry Rowlingson.

glmmBUGS Generalized Linear Mixed Models with WinBUGS. By Patrick Brown.

glmnet Extremely efficient procedures for fitting the entire lasso or elastic-net regularization path for linear regression, logistic and multinomial regression models, using cyclical coordinate descent in a pathwise fashion. By Jerome Friedman, Trevor Hastie and Rob Tibshirani.

gmm Estimation of parameters using the generalized method of moments (GMM) of Hansen (1982). By Pierre Chausse.

grade Binary Grading functions for R. By Leif Johnson.

gtm Generative topographic mapping. By Ondrej Such.

hlr Hidden Logistic Regression. Implements the methods described in Rousseeuw and Christman (2003) to cope with separation issues and outliers in logistic regression. Original S-PLUS code by Peter J. Rousseeuw and Andreas Christmann, R port by Tobias Verbeke.

hwriter HTML Writer: easy-to-use and versatile functions to output R objects in HTML format. By Gregoire Pau.

ic.infer Inequality constrained inference in linear normal situations. Implements parameter estimation in normal (linear) models under linear equality and inequality constraints as well as likelihood ratio tests involving inequality-constrained hypotheses. By Ulrike Groemping.

ic50 Standardized high-throughput evaluation of compound screens. Calculation of IC50 values, automatic drawing of dose-response curves and validation of compound screens on 96- and 384-well plates. By Peter Frommolt.

icomp Calculates the ICOMP criterion and its variations. By Jake Ferguson.

imputeMDR Multifactor Dimensionality Reduction (MDR) analysis for incomplete data. By Junghyun Namkung, Taeyoung Hwang, Min-Seok Kwon, Sunggon Yi and Wonil Chung.

intervals Tools for working with and comparing sets of points and intervals. By Richard Bourgon.

kml *K*-Means for Longitudinal data (kml), a non parametric algorithm for clustering longitudinal data. By Christophe M. Genolini.

laercio Functions to compare group means (Duncan test, Tukey test and Scott-Knott test). By Laercio Junio da Silva.

lda.cv Cross-validation for linear discriminant analysis. By Wenxuan Zhong.

lmom L-moments: computation of L-moments of distributions and data samples, parameter estimation, L-moment ratio diagram, and plot against quantiles of an extreme-value distribution. By J. R. M. Hosking.

loglognorm Double log normal distribution functions. By Heike Trautmann, Detlef Steuer and Olaf Mersmann.

lpSolveAPI An R interface to the lp_solve library API. lp_solve is a Mixed Integer Linear Programming (MILP) solver with support for pure linear, (mixed) integer/binary, semi-continuous and special ordered sets (SOS) models. By Kjell Konis.

lpc Implements the Lassoed Principal Components (LPC) method of Witten & Tibshirani (2008) for identification of significant genes in a microarray experiment. By Daniela M Witten and Robert Tibshirani.

marelac Datasets, chemical and physical constants and functions, routines for unit conversions, etc, for the marine and lacustrine sciences. By Karline Soetaert and Filip Meysman.

marginalmodelplots Marginal model plots for linear and generalized linear models, including tools for bandwidth exploration. By Andrew Redd.

mco Functions for multi criteria optimization using genetic algorithms and related test problems. By Heike Trautmann, Detlef Steuer and Olaf Mersmann.

mi Missing-data imputation and model checking. By Andrew Gelman, Jennifer Hill, Masanao Yajima, Yu-Sung Su and Maria Grazia Pittau.

mirf Multiple Imputation and Random Forests for unobservable phase, high-dimensional data. Applies a combination of missing haplotype imputation via the EM algorithm of Excoffier and Slatkin (1995) and modeling trait-haplotype associations via the Random Forest algorithm, as described in "Multiple imputation and random forests (MIRF) for unobservable high-dimensional data" by B.A.S. Nonyane and A.S. Foulkes (2007, The International Journal of Biostatistics 3(1): Article 12). By Yimin Wu, B. Aletta S. Nonyane and Andrea S. Foulkes.

mixer Estimation of Erdős-Rényi mixture for graphs. By Christophe Ambroise, Gilles Grasseau, Mark Hoebeke and Pierre Latouche.

mixlow Assessing drug synergism/antagonism. By John Boik.

mlogit Estimation of the multinomial logit model with alternative and/or individual specific variables. By Yves Croissant.

mpm Multivariate Projection Methods: exploratory graphical analysis of multivariate data, specifically gene expression data with different projection methods: principal component analysis, correspondence analysis, spectral map analysis. By Luc Wouters.

mrdrc Model-Robust Concentration-Response Analysis semi-parametric modeling of continuous and quantal concentration/dose-response data. By Christian Ritz and Mads Jeppe Tarp-Johansen.

msBreast A dataset of 96 protein mass spectra generated from a pooled sample of nipple aspirate fluid (NAF) from healthy breasts and breasts with cancer. By Lixin Gong, William Constantine and Yu Alex Chen.

msDilution A dataset of 280 MALDI-TOF mass spectra generated from a dilution experiment aimed at elucidating which features in MALDI-TOF mass spectrometry data are informative for quantifying peptide content. By Lixin Gong, William Constantine and Yu Alex Chen.

msProstate A dataset of protein mass spectra generated from patients with prostate cancer, benign prostatic hypertrophy, and normal controls. By Lixin Gong, William Constantine and Alex Chen.

nparcomp Computation of nonparametric simultaneous confidence intervals and simultaneous p-values for relative contrast effects in the unbalanced one way layout. The simultaneous confidence intervals can be computed using multivariate normal distribution, multivariate t-distribution with a Satterthwaite Approximation of the degree of freedom, or using multivariate range preserving transformations with logit or probit as transformation function. By Frank Konietschke.

onemap Software for constructing genetic maps in outcrossing species. Analysis of molecular marker data from non-model systems to simultaneously estimate linkage and linkage phases. By Gabriel Rodrigues Alves Margarido, with contributions from Antonio Augusto Franco Garcia.

opentick Provide an interface to opentick real time and historical market data. By Josh Ulrich.

orloca The Operations Research LOCational Analysis models. Deals with the min-sum or center location problems. By Fernando Fernandez-Palacin and Manuel Munoz-Marquez.

orloca.es Spanish version of package **orloca**. By Fernando Fernandez-Palacin and Manuel Munoz-Marquez.

orth Performs multivariate logistic regressions by way of orthogonalized residuals. As a special case, the methodology recasts alternating logistic regressions in a way that is consistent with standard estimating equation theory. Cluster diagnostics and observation level diagnostics such as leverage and Cook's distance are computed based on an approximation. By Kunthel By, Bahjat F. Qaqish and John S. Preisser.

pack Functions to easily convert data to binary formats other programs/machines can understand. By Josh Ulrich.

packClassic Illustration of the tutorial "S4: From Idea To Package". By Christophe Genolini and some reader that sent useful comments.

paltran Functions for paleolimnology: wa-regression (see also package **analogue**), wa-pls and MW regression. By Sven Adler.

pec Prediction Error Curves for survival models: validation of predicted surival probabilities using inverse weighting and resampling. By Thomas A. Gerds.

peperr Parallelized Estimation of Prediction Error. Prediction error estimation through resampling techniques, possibly accelerated by parallel execution on a computer cluster. By Christine Porzelius and Harald Binder.

picante Phylocom integration, community analyses, null-models, traits and evolution in R. By Steve Kembel, David Ackerly, Simon Blomberg, Peter Cowan, Matthew Helmus and Cam Webb.

plyr Tools for splitting, applying and combining data. By Hadley Wickham.

qdg QTL Directed Graphs: infer QTL-directed dependency graphs for phenotype networks. By Elias Chaibub Neto and Brian S. Yandell.

qlspack Quasi Least Squares (QLS) package. QLS is a two-stage computational approach for estimation of the correlation parameters within the framework of GEE. It helps solving parameters in mean, scale, and correlation structures for longitudinal data. By Jichun Xie and Justine Shults.

rPorta An R interface to a modified version of PORTA (see `http://www.zib.de/Optimization/Software/Porta/`). By Robin Nunkesser, Silke Straatmann and Simone Wenzel.

randtoolbox Toolbox for pseudo and quasi random number generation and RNG tests. Provides general linear congruential generators (Park Miller) and multiple recursive generators (Knuth TAOCP), generalized feedback shift register (SF-Mersenne Twister algorithm and WELL generator), the Torus algorithm, and some additional tests (gap test, serial test, poker test, ...). By Christophe Dutang, with the SFMT algorithm from M. Matsumoto and M. Saito, the WELL generator from P. L'Ecuyer, and the Knuth-TAOCP RNG from D. Knuth).

rbounds Perform Rosenbaum bounds sensitivity tests for matched data. Calculates bounds for binary data, Hodges-Lehmann point estimates, Wilcoxon signed-rank test, and for data with multiple matched controls. Is designed to work with package **Matching**. By Luke J. Keele.

rdetools Functions for Relevant Dimension Estimation (RDE) of a data set in feature spaces, applications to model selection, graphical illustrations and prediction. By Jan Saputra Mueller.

repolr Repeated measures proportional odds logistic regression via generalized estimating equations. By Nick Parsons.

rjags Bayesian graphical models via an interface to the JAGS MCMC library. By Martyn Plummer.

rootSolve Nonlinear root finding, equilibrium and steady-state analysis of ordinary differential equations. Includes routines that: (1) generate gradient and Jacobian matrices (full and banded), (2) find roots of non-linear equations by the Newton-Raphson method, (3) estimate steady-state conditions of a system of (differential) equations in full, banded or sparse form, using the Newton-Raphson method, or by dynamically running, (4) solve the steady-state conditions for uni-and multicomponent 1-D and 2-D reactive transport models (boundary value problems of ODE) using the method of lines approach. By Karline Soetaert.

roxygen Literate Programming in R: a Doxygen-like in-source documentation system for Rd, collation, namespace and callgraphs. By Peter Danenberg and Manuel Eugster.

rscproxy Provides a portable C-style interface to R (StatConnector) used by third party applications, most notable, but not limited to, rcom/R Scilab (D)COM Server, ROOo and other systems. By Thomas Baier.

runjags Run Bayesian MCMC models in the BUGS syntax from within R. Includes functions to read external WinBUGS type text files, and allows several ways of automatically specifying model data from existing R objects or R functions. Also includes functions to automatically calculate model run length, autocorrelation and Gelman Rubin statistic diagnostics for all models to simplify the process of achieving chain convergence. Requires Just Another Gibbs Sampler (JAGS, `http://www-fis.iarc.fr/~martyn/software/jags/`) for most functions. By Matthew Denwood (funded as part of the DEFRA VTRI project 0101).

rwm APL-like functions for managing R workspaces. By A.I. McLeod.

sdtalt Signal detection theory measures and alternatives, as detailed in the book "Functions for traditional and multilevel approaches to signal detection theory" by D.B. Wright, R. Horry and E.M. Skagerberg, E.M. (in press, Behavior Research Methods). By Daniel B. Wright.

sensR Thurstonian models for sensory discrimination. By Rune Haubo B Christensen and Per Bruun Brockhoff.

singlecase Various functions for the single-case research in neuropsychology, mainly dealing with the comparison of a patient's test score (or score difference) to a control or normative sample. These methods also provide a point estimate of the percentage of the population that would obtain a more extreme score (or score difference) and, for some problems, an accompanying interval estimate (i.e., confidence limits) on this percentage. By Matthieu Dubois.

smacof provides the following approaches to multidimensional scaling (MDS) based on stress minimization by means of majorization (smacof): Simple smacof on symmetric dissimilarity matrices, smacof for rectangular matrices (unfolding models), smacof with constraints

on the configuration, three-way smacof for individual differences (including constraints for idioscal, indscal, and identity), and spherical smacof (primal and dual algorithm). Each of these approaches is implemented in a metric and nonmetric manner including primary, secondary, and tertiary approaches for tie handling. By Jan de Leeuw and Patrick Mair.

snowfall Top-level wrapper around **snow** for easier development of parallel R programs. All functions work in sequential mode, too, if no cluster is present or wished. By Jochen Knaus.

spssDDI Read SPSS System files and produce valid DDI version 3.0 documents. By Guido Gay.

ssize.fdr Sample size calculations for microarray experiments, featuring appropriate sample sizes for one-sample *t*-tests, two-sample *t*-tests, and *F*-tests for microarray experiments based on desired power while controlling for false discovery rates. For all tests, the standard deviations (variances) among genes can be assumed fixed or random. This is also true for effect sizes among genes in one-sample experiments and differences in mean treatment expressions for two-sample experiments. Functions also output a chart of power versus sample size, a table of power at different sample sizes, and a table of critical test values at different sample sizes. By Megan Orr and Peng Liu.

tframePlus Time Frame coding kernel extensions. By Paul Gilbert.

tileHMM Hidden Markov Models for ChIP-on-Chip Analysis. Provided parameter estimation methods include the Baum-Welch algorithm and Viterbi training as well as a combination of both. By Peter Humburg.

timereg Programs for the book "Dynamic Regression Models for Survival Data" by T. Martinussen and T. Scheike (2006, Springer Verlag), plus more recent developments. By Thomas Scheike, with contributions from Torben Martinussen and Jeremy Silver.

tis Functions and S3 classes for time indexes and time indexed series, which are compatible with FAME frequencies. By Jeff Hallman.

topmodel An R implementation of TOPMODEL, based on the 1995 FORTRAN version by Keith Beven. Adapted from the C translation by Huidae Cho. By Wouter Buytaert.

tossm Testing Of Spatial Structure Methods. Provides a framework under which methods for using genetic data to detect population structure and define management units can be tested. By Karen Martien, Dave Gregovich and Mark Bravington.

treelet Treelet: a novel construction of multi-scale bases that extends wavelets to non-smooth signals. Returns a hierarchical tree and a multi-scale orthonormal basis which both reflect the internal structure of the data. By Di Liu.

tsModel Time Series Modeling for Air Pollution and Health. By Roger D. Peng, with contributions from Aidan McDermott.

uncompress Functionality for decompressing '.Z' files. By Nicholas Vinen.

waveclock Time-frequency analysis of cycling cell luminescence data. Provides function `waveclock` designed to assess the period and amplitude of cycling cell luminescence data. The function reconstructs the modal frequencies from a continuous wavelet decomposition of the luminescence data using the "crazy climbers" algorithm described in the book "Practical Time-Frequency Analysis: Gabor and Wavelet Transforms with an Implementation in S" by René Carmona, Wen L. Hwang and Bruno Torresani (1998, Academic Press). By Tom Price.

wgaim Whole Genome Average Interval Mapping for QTL detection using mixed models. Integrates sophisticated mixed modeling methods with a whole genome approach to detecting significant QTLs in linkage maps. By Julian Taylor, Simon Diffey, Ari Verbyla and Brian Cullis.

Other changes

- Package **PARccs** was renamed to **pARccs**.
- Package **BRugs** was moved to the Archive (available via CRANextras now).
- Package **HighProbability** was moved to the Archive (now contained in bundle **empiricalBayes**).
- Package **brlr** was moved to the Archive.
- Package **paleoTSalt** was moved to the Archive (integrated in package **paleoTS**).
- Package **torus** was moved to the Archive (replaced by package **randtoolbox**).
- Packages **data.table** and **elasticnet** were resurrected from the Archive.

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

News from the Bioconductor Project

Bioconductor Team
Program in Computational Biology
Fred Hutchinson Cancer Research Center

We are pleased to announce Bioconductor 2.3, released on October 22, 2008. Bioconductor 2.3 is compatible with R 2.8.0, and consists of 294 packages. There are 37 new packages, and enhancements to many others. Explore Bioconductor at `http://bioconductor.org`, and install standard or individual packages with

```
> source("http://bioconductor.org/biocLite.R")
> biocLite() # install standard packages...
> biocLite("rtracklayer") # or rtracklayer
```

New packages

New to this release are powerful packages for diverse areas of high-throughput analysis. Highlights include:

Assay technologies such as HELP and MeDIP DNA methylation (**HELP**, **MEDME**), micro-RNAs (**microRNA**, **miRNApath**), and array comparative genomic hybridization (**ITALICS**, **CGHregions**, **KCsmart**).

Pathways and integrative analysis packages such as **SIM** (gene set enrichment for copy number and expression data), **domainsignatures** (InterPro gene set enrichment), **minet** (mutual information network inference) and **miRNApath** (pathway enrichment for micro-RNA data).

Interoperability with the ArrayExpress data base (**ArrayExpress**), the Chemmine chemical compound data base (**ChemmineR**), the PSI-MI protein interaction data base (**RpsiXML**) and external visualization tools such as the X:Map exon viewer (**xmapbridge**).

Algorithms for machine learning, such as **BicARE**, **dualKS**, **CMA**) and iterative Bayesian model averaging (**IterativeBMA**, **IterativeBMAsurv**).

Refined expression array methods including preprocessing (e.g., **multiscan**), contaminant and outlier detection (**affyContam**, **arrayMvout**, **parody**), and small-sample and other assays for differential expression (e.g., **logitT**, **DFP**, **LPEadj**, **PLPE**).

Annotations

Bioconductor 'Annotation' packages contain biological information about microarray probes and the genes they are meant to interrogate, or contain ENTREZ gene-based annotations of whole genomes. This release marks the completion of our transition from environment-based to SQLite-based annotation packages. SQLite annotation packages allow for efficient memory use and facilitate more complicated data queries. The release now supports a menagerie of 15 different model organisms, from Arabidopsis to Zebrafish, nearly doubling the number of species compared to the previous Bioconductor release. We have also increased the amount of information in each annotation package; the inst/NEWS file in **AnnotationDbi** provides details.

High-throughput sequencing

An ensemble of new or expanded packages introduces tools for 'next generation' DNA sequence data. This data is very large, consisting of 10s to 100s of millions of 'reads' (each 10s to 100s of nucleotides long), coupled with whole genome sequences (e.g., 3 billion nucleotides in the human genome). **BSgenome** provides a foundation for representing whole-genome sequences; there are 13 model organisms represented by 18 distinct genome builds, and facilities for users to create their own genome packages. Functionality in the **Biostrings** package performs fast or flexible alignments between reads and genomes. **ShortRead** provides tools for import and exploration of common data formats. **IRanges** offers an emerging infrastructure for representing very large data objects, and for range-based representations. The **rtracklayer** package interfaces with genome browsers and their track layers. **HilbertVis** and **HilbertVisGUI** provide an example of creative approaches to visualizing this data, using space-filling (Hilbert) curves that maintain, as much as possible, the spatial information implied by linear chromosomes.

Other activities

Bioconductor package maintainers and the Bioconductor team invest considerable effort in producing high-quality software. New packages are reviewed on both technical and scientific merit, before being added to the 'development' roster for the next release. Release packages are built daily on Linux, Windows, and Mac platforms, tracking the most recent released versions of R and the packages hosted on CRAN repositories. The active Bioconductor mailing lists (`http://bioconductor.org/docs/mailList.html`) connect users with each other, to domain experts, and to maintainers eager to ensure that

their packages satisfy the needs of leading edge approaches. The Bioconductor community meets at our annual conference in Seattle; well over 100 individuals attended the July meeting for a combination of scientific talks and hands-on tutorials.

Looking forward

This will be a dynamic release cycle. New contributed packages are already under review, and our build machines have started tracking the latest development versions of R. It is hard to know what the future holds, but past experience points to surprise — at the creativity, curiosity, enthusiasm, and commitment of package developers, and at the speed of technological change and growth of data. In addition to development of high-quality algorithms to address microarray data analysis, we anticipate continued efforts to leverage diverse external data sources and to meet the challenges of presenting high volume data in rich graphical contexts.

High throughput sequencing technologies represent a particularly likely area for development. First-generation approaches with relatively short reads, restricted application domains, and small numbers of sample individuals are being supplanted by newer technologies producing longer and more numerous reads. New protocols and the intrinsic curiosity of biologists are expanding the range of questions being addressed, and creating a concomitant need for flexible software analysis tools. The increasing affordability of high throughput sequencing technologies means that multi-sample studies with non-trivial experimental designs are just around the corner. The statistical analytic abilities and flexibility of R and Bioconductor represent ideal tools for addressing these challenges.

useR! 2008

The international R user conference 'useR! 2008' took place at the Technische Universität Dortmund, Dortmund, Germany, August 12-14, 2008.

This world-wide meeting of the R user community focussed on

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

The program comprised invited lectures, user-contributed sessions and pre-conference tutorials.

More than 400 participants from all over the world met in Dortmund and heard more than 170 talks. It was a pleasure for us to see that the lecture rooms were available after heavy weather and some serious flooding of the building two weeks before the conference. People were carrying on discussions the whole time: before sessions, during the coffee and lunch breaks, and during the social events such as

- the reception in the Dortmund city hall by the Mayor (who talked about the history, present, and future of Dortmund), and
- the conference dinner in the 'Signal Iduna Park' stadium, the famous home ground of the football (i.e. soccer) team 'Borussia Dortmund'. Buses brought the participants to the stadium where a virtual game was going on: at least, noise was played, made by the more than 80000 supporters who regularly attend games in the stadium. After the dinner, Marc Schwartz got the birthday cake he really deserves for his work in the R community.

It was a great plesure to meet so many people from the whole R community in Dortmund, people who formerly knew each other only from 'virtual' contact by email and had never met in person.

Invited lectures

Those who attended had the opportunity to listen to several talks. Invited speakers talked about several hot topics, including:

- *Peter Bühlmann*: Computationally Tractable Methods for High-Dimensional Data
- *John Fox* and *Kurt Hornik*: The Past, Present, and Future of the R Project, a double-feature presentation including
  Social Organization of the R Project (John Fox),
  Development in the R Project (Kurt Hornik)
- *Andrew Gelman*: Bayesian Generalized Linear Models and an Appropriate Default Prior
- *Gary King*: The Dataverse Network
- *Duncan Murdoch*: Package Development in Windows
- *Jean Thioulouse*: Multivariate Data Analysis in Microbial Ecology – New Skin for the Old Ceremony
- *Graham J. Williams*: Deploying Data Mining in Government – Experiences With R/Rattle

User-contributed sessions

User contributed sessions brought together R users, contributers, package maintainers and developers in the S spirit that 'users are developers'. People from different fields showed us how they solve problems with R in fascinating applications. The scientific program was organized by members of the program committee, including *Micah Altman, Roger Bivand, Peter Dalgaard, Jan de Leeuw, Ramón Díaz-Uriarte, Spencer Graves, Leonhard Held, Torsten Hothorn, François Husson, Christian Kleiber, Friedrich Leisch, Andy Liaw, Martin Mächler, Kate Mullen, Ei-ji Nakama, Thomas Petzoldt, Martin Theus*, and *Heather Turner*, and covered topics such as

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

Talks in Kaleidoscope sessions presented interesting topics to a broader audience, while talks in Focus sessions led to intensive discussions on the more focussed topics. As was the case at former useR! conferences, the participants listened to talks the whole time; it turned out that the organizers underestimated the interest of the participants for some sessions and chose lecture rooms that were too small to accommodate all who wanted to attend.

Pre-conference tutorials

Before the start of the official program, half-day tutorials were offered on Monday, August 11.
In the morning:

- *Douglas Bates*: Mixed Effects Models
- *Julie Josse, François Husson, Sébastien Lê*: Exploratory Data Analysis
- *Martin Mächler, Elvezio Ronchetti*: Introduction to Robust Statistics with R
- *Jim Porzak*: Using R for Customer Segmentation
- *Stefan Rüping, Michael Mock, and Dennis Wegener*: Distributed Data Analysis Using R
- *Jing Hua Zhao*: Analysis of Complex Traits Using R: Case studies

In the afternoon:

- *Karim Chine*: Distributed R and Bioconductor for the Web
- *Dirk Eddelbuettel*: An Introduction to High-Performance R
- *Andrea S. Foulkes*: Analysis of Complex Traits Using R: Statistical Applications
- *Virgilio Gómez-Rubio*: Small Area Estimation with R
- *Frank E. Harrell, Jr.*: Regression Modelling Strategies
- *Sébastien Lê, Julie Josse, François Husson*: Multiway Data Analysis
- *Bernhard Pfaff*: Analysis of Integrated and Cointegrated Time Series

At the end of the day, Torsten Hothorn and Fritz Leisch opened the Welcome mixer by talking about the close relationship among *R*, the useR! organizers, and Dortmund.

More information

A web page offering more information on 'useR! 2008' is available at:
`http://www.R-project.org/useR-2008/`.
That page now includes slides from many of the presentation as well as some photos of the event. Those who could not attend the conference should feel free to browse and see what they missed, as should those who attended but were not able to make it to all of the presentations that they wanted to see.

The organizing committee:
Uwe Ligges, Achim Zeileis, Claus Weihs, Gerd Kopp, Friedrich Leisch, and Torsten Hothorn
`useR-2008@R-project.org`

Forthcoming Events: useR! 2009

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

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

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

The program comprises invited lectures, usercontributed sessions and pre-conference tutorials.

Invited lectures

R has become the standard computing engine in more and more disciplines, both in academia and the business world. How R is used in different areas will be presented in hot topics as evidenced by the list of the invited speakers: *Adele Cutler, Peter Dalgaard, Jerome H. Friedman, Michael Greenacre, Trevor Hastie, John Storey* and *Martin Theus*.

User-contributed sessions

The sessions will be a platform to bring together R users, contributers, package maintainers and developers in the S spirit that 'users are developers'. People from different fields will show us how they solve problems with R in fascinating applications. The scientific program will include *Douglas Bates, Vincent Carey, Arthur Charpentier, Pierre-André Cornillon, Nicolas Hengartner, Torsten Hothorn, François Husson, Jan de Leeuw, Friedrich Leisch, Ching Fan Sheu* and *Achim Zeileis*. The program will cover topics of current interest such as

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

Pre-conference tutorials

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

Call for papers

We invite all R users to submit abstracts on topics presenting innovations or exciting applications of R. A web page offering more information on 'useR!2009' is available at:

`http://www.R-project.org/useR-2009/`

Abstract submission and registration will start in November 2008.

The organizing committee *David Causeur, Julie Josse, François Husson, Maela Kloareg, Sébastien Lê, Eric Matzner-Løber* and *Jérôme Pagès* will be very happy to meet you in Rennes, France, the capital of Brittany, famous for its architecture (the Parlement de Bretagne, private mansions, ...), its night life (Festival des tombées de la nuit) and its cuisine (the Festival Gourmand, the Lices Market, ...). `useR-2009@R-project.org`

Forthcoming Events: DSC 2009

Hot on the heels of 'useR! 2009', the sixth international workshop on Directions in Statistical Computing (DSC 2009) will take place at the Center for Health and Society, University of Copenhagen, Denmark, 13th-14th of July 2009.

The workshop will focus on, but is not limited to, open source statistical computing and aims to provide a platform for exchanging ideas about developments in statistical computing (rather than 'only' the usage of statistical software for applications).

Traditionally, this is a workshop with a flatter structure than the useR! series, mainly designed for formal and informal discussions among developers of statistical software. Consequently there are no invited speakers, although the programme committee will select some talks for plenary presentations.

Call for papers

It is expected that a large proportion of participants will bring a presentation. We particularly welcome contributions advancing statistical computing as an independent topic. Abstracts can be submitted via the website once it becomes fully operational.

A web page offering more information on 'DSC 2009' is available at:

`http://www.R-project.org/dsc-2009/`

Abstract submission and registration will start in November 2008.

The organizing committee, *Peter Dalgaard, Claus Ekstrøm, Klaus K. Holst* and *Søren Højsgaard*, will be very happy to meet you in Copenhagen at the height of Summer. When the weather is good, it is extremely beautiful here, not too hot and with long summer evenings (it might also rain for days on end, though).

`dsc2009@biostat.ku.dk`

R Foundation News

by Kurt Hornik

Donations and new members

New benefactors

REvolution Computing, USA

New supporting members

Peter Ehlers, Canada
Chel Hee Lee, Canada

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