The R Journal

Volume 5/2, December 2013

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

Contents

Editorial

by Hadley Wickham

Welcome to volume 5, issue 2 of The R Journal. I'm very pleased to include 21 articles about R for your enjoyment.

The end of the year also brings changes to the editorial board. Martyn Plummer is leaving the board after four years. Martyn was responsible for writing up standard operating procedures for journal, an act which has made my life as a new editor considerably easier! We welcome Michael Lawrence, who will join the editorial board in 2014. I am stepping down as Editor-in-Chief and will be leaving this task in the capable hands of Deepayan Sarkar.

Hadley Wickham, RStudio
hadley.wickham@r-project.org

factorplot: Improving Presentation of Simple Contrasts in Generalized Linear Models

by David A. Armstrong II

Abstract Recent statistical literature has paid attention to the presentation of pairwise comparisons either from the point of view of the reference category problem in generalized linear models (GLMs) or in terms of multiple comparisons. Both schools of thought are interested in the parsimonious presentation of sufficient information to enable readers to evaluate the significance of contrasts resulting from the inclusion of qualitative variables in GLMs. These comparisons also arise when trying to interpret multinomial models where one category of the dependent variable is omitted as a reference. While considerable advances have been made, opportunities remain to improve the presentation of this information, especially in graphical form. The **factorplot** package provides new functions for graphically and numerically presenting results of hypothesis tests related to pairwise comparisons resulting from qualitative covariates in GLMs or coefficients in multinomial logistic regression models.

Introduction

The problem of presenting information about categorical covariates in generalized linear models is a relatively simple one. Nevertheless, it has received some attention in the recent literature. To be clear about the problem, consider the following linear model where y is the dependent variable and $G = \{1, 2, \ldots, m\}$ is a categorical independent variable that can be represented in the regression model by $m - 1$ dummy regressors, each one representing a different category of G. The reference category, of course, is omitted. Thus, the model looks as follows:

$$E(y_i) = \mu_i \tag{1}$$
$$g(\mu_i) = \beta_0 + \beta_1 D_{i1} + \beta_2 D_{i2} + \cdots + \beta_{m-1} D_{im-1} + \beta_m X_{i1} + \ldots + \beta_{m+k-1} X_{ik} + \varepsilon_i, \tag{2}$$

where $D_{i1} = 1$ if $G_i = 1$, $D_{i2} = 1$ if $G_i = 2$, etc. X_{ik} represent an arbitrary set of additional variables of any type. Here, each of the coefficients on the dummy regressors for G $(\beta_1, \ldots, \beta_{m-1})$ gives the difference in the conditional transformed mean of y between the category represented by the dummy regressor and the reference category, controlling for all of the other X_{ik}. However, the $m - 1$ coefficients for the categories of G imply $\frac{m(m-1)}{2}$ simple contrasts representing every pairwise comparison between categories of G. Any single pairwise comparison of non-reference category coefficients can be conducted in a straightforward fashion. If the goal is to discern whether the conditional mean of y given $G = 1$ is different from the conditional mean of y given $G = 2$ holding all of the X variables constant, the quantity of interest is:

$$t = \frac{b_1 - b_2}{\sqrt{V(b_1 - b_2)}}, \tag{3}$$

where

$$V(b_1 - b_2) = V(b_1) + V(b_2) - 2V(b_1, b_2). \tag{4}$$

Thus, the calculation is not difficult, but calculating and presenting all of these differences can become cumbersome, especially as m gets large.[1] The problem comes not in the calculation of these quantities, but in the parsimonious presentation of this information that will allow users to evaluate any desired (simple) contrasts easily. Below, I discuss two extant methods used to present such information. Floating absolute risk (FAR) was first suggested by Easton et al. (1991) and was more rigorously justified, though with different estimation strategies, by Firth and De Menezes (2004); de Menezes (1999); Plummer (2004). FAR is a means of overcoming the reference category problem by calculating floating variances for all levels of a factor (including the reference category). These floating variances can be used to perform hypothesis tests or construct floating confidence intervals that facilitate the graphical comparison of different categories (i.e., [log-]relative risks). The multiple comparisons literature has traditionally been focused on finding the appropriate p-values to control either the

[1] Tools to carry out these computations already exist in the **multcomp** package in R (Hothorn et al., 2008).

family-wise error rate (e.g., Holm, 1979) or the false discovery rate (e.g., Benjamini and Hochberg, 1995) in a set of simultaneous hypothesis tests. Presentation of this information has either been in the form of line displays (e.g., Steel and Torrie, 1980) or compact letter displays (e.g., Gramm et al., 2006) with more recent innovation here by Graves et al. (2012).

However, when simple contrasts are the only quantities of interest, neither method above is perfect. When floating/quasi-variances are presented, the user still has to evaluate a potentially large number of hypothesis tests by either relying on the overlap in the floating confidence intervals or by calculating the floating t-statistic. Either solution requires a good deal of cognitive energy on the part of the analyst or reader. Compact letter displays do well at identifying patterns of statistical significance, but are perhaps cumbersome to investigate when patterns of (in)significance are complicated and, though mitigated to some degree, the problem still exists for the more recent multcompTs suggested by Graves et al. (2012). Below, I discuss a means for presenting this information in a manner that will permit the immediate evaluation of all the $m(m-1)/2$ hypothesis tests associated with simple contrasts. The method I propose can also calculate analytical standard errors that are not prone to the same potential inferential errors produced by floating variances. I provide methods to summarize, print and plot the information in a way that is both visually appealing and straightforward to understand.

Solutions to the reference category problem

There are a number of reasonable solutions to the reference category problem.[2] The first solution is to present all of the covariance information required to calculate t-statistics for contrasts of interest (i.e., the variance-covariance matrix of the estimators). This solution provides the reader with all necessary information to make inferences. However, it does not provide an easy way for all of these inferences to be presented. Another solution is to re-estimate the model with different reference categories in turn.[3] This method produces the correct inferential information, but it is inelegant. The modal response to the reference category problem is a failure to do anything to discover (or allow readers to investigate) the implied pairwise differences not captured by the estimated coefficients.

Easton et al. (1991) proposed the idea of floating absolute risk as a means for evaluating multiple comparisons in matched case-control studies. The idea was to provide sufficient information such that readers could perform multiple comparisons with estimates of floating absolute risk at the expense of presenting a single extra number for each binary variable representing a level of a categorical covariate (i.e., risk factor). Although Greenland et al. (1999) disagreed on terminology and on the utility of Easton's idea of a floating scale, they agreed on the utility presenting information that would permit users to easily make the right inferences about relative risks among any levels of a categorical risk factor. Both Firth and De Menezes (2004) and Plummer (2004) provided a more rigorous statistical foundation on which to build estimates of floating absolute risk (or as Firth and De Menezes call them, quasi-variances). Firth and De Menezes' method has been operationalized in R in the **qvcalc** package (Firth, 2010) and both the methods of Plummer as well as Greenland et al. have been operationalized in the `float()` and `ftrend()` functions, respectively, in the **Epi** package (Carstensen et al., 2013). In general, these solutions allow sufficient (or nearly sufficient) information to be presented in a single column of a statistical table that makes valid, arbitrary multiple comparisons possible.

The measures of floating absolute risk are often used to create floating (or quasi-) confidence intervals.[4] Presenting these intervals allows the user to approximately evaluate hypothesis tests about any simple contrast. While the exact nature of these confidence intervals is somewhat controversial (for a discussion, see Easton and Peto (2000); Greenland et al. (1999, 2000)), all agree that confidence intervals can be profitably put around some quantity (either the log-relative risks versus the reference category or the floating trend) to display the uncertainty around these quantities and permit visual hypothesis tests.

The methods discussed above still require the analyst or reader to either evaluate the pairwise hypothesis tests based on the extent to which confidence intervals overlap or calculate the floating t-statistic for each desired contrast. If the former, readers must still engage in a cognitive task of position detection (Cleveland, 1985) and then make an inference based on the extent to which intervals overlap. As the horizontal distance between vertically-oriented floating confidence intervals grows, this task becomes more difficult. Finally, as Easton et al. (1991) suggests, floating variances are a

[2]The problem here applies particularly to polytomous, unordered risk factors or covariates. The case of ordinal risk factors, where only the difference in adjacent categories is of interest, is a bit less troublesome and will not be dealt with separately here.

[3]In fact, this re-parameterization method could be used to deal with more complicated contrasts, too. For example, it could be used to deal with the problem proposed by Greenland et al. (1999) wherein they wanted to estimate the relative risk of being above a particular category on birthweight.

[4]Occasionally, quasi-variance estimates are negative, which provide the right inferences, but do not permit plotting of quasi-confidence intervals.

"virtually sufficient" summary of the uncertainty relating to relative risks; however, they can produce erroneous inferences if the error rate is sufficiently high. Both Firth and De Menezes (2004) and Plummer (2004) provide methods for calculating this error rate, which is often small relative to other sources of error in the model.

To put a finer point on the problem, consider the example below using data from Ornstein (1976) from the **car** package (Fox and Weisberg, 2011). The model of interest is:

$$\begin{aligned} \text{Interlocks}_i &\sim \text{Poisson}(\mu_i) \\ \log(\mu_i) &= \beta_0 + \beta_1 \log_2(\text{Assets}_i) + \gamma \text{Sector}_{ij} + \boldsymbol{\theta}\text{Nation}_{im} \end{aligned} \tag{5}$$

where γ represents a set of coefficients on the $j = 9$ non-reference category dummy variables for the 10 sectors represented in the data and $\boldsymbol{\theta}$ is the set of coefficients for the $m = 3$ coefficients on the non-reference category dummy variables representing the four nations in the dataset. The goal is to determine which sectors (and/or nations) have significantly different transformed conditional means of *Interlocks*. The quasi-variances can be presented along with the coefficients permitting hypothesis testing at the reader's discretion. This approach is economical, but still requires the interested reader to make 27 pairwise hypothesis tests for sector and three pairwise hypothesis tests for nation, beyond those presented in the coefficient table.

The plot of the floating confidence intervals provides similar information, but readers are still required to make judgements about statistical significance with a visual method prone to occasional inferential errors. Consider Figure 1, which presents confidence intervals using the three different functions that produce floating variances R — `qvcalc()`, `float()` and `ftrend()`.[5] In the figure, the floating confidence interval for the mining sector overlaps four other floating confidence intervals and does not overlap the remaining five intervals.[6] Advice from Smith (1997) suggests that only confidence intervals not containing the point estimate against which the test is being done are significant. Here, all of the pairwise differences with the mining coefficient are significant because none of the point estimates are within the 95% confidence interval for mining. A more conservative strategy is to fail to reject null hypotheses where confidence intervals overlap and to reject otherwise. Using this criterion, the mining sector is different from five other coefficients — Agriculture, Banking, Construction, Finance and Wood. Browne (1979) shows that making inferences from confidence intervals requires a knowledge of the different sampling variances of the underlying random variables for which the confidence intervals have been constructed (i.e., the widths of the intervals matter); the decision does not rest solely on the extent to which the intervals overlap. While Browne's method may produce more appropriate inferences, it is hardly less work than producing the hypothesis tests directly. When the appropriate pairwise hypothesis tests are performed, without adjusting the p-values for multiple testing, it is clear that the mining coefficient is different from seven coefficients when using a two-sided test, as Table 1 shows.

Even if the evidence regarding the outcome of a hypothesis test from two confidence intervals is clear, there are other potential sources of error. Cleveland (1985) finds that detecting position along a common scale is one of the easiest tasks of graphical perception, but that discerning length is considerably more difficult. His experiments show that readers are prone to errors in even the easiest graphical perception tasks and the error rate is nearly twice as high when readers are asked to adjudicate the relative lengths of lines. Conducting hypothesis tests using confidence intervals is an endeavor rife with opportunities for inferential errors.

Means for calculating and presenting models with multiple simple contrasts have developed in the multiple testing literature as well. While the thrust of the literature mentioned above was dealing with the reference category problem directly, the multiple comparisons literature has placed greater focus on finding the appropriate p-values for a set of hypothesis tests rather than a single test. This can be accomplished through controlling the family-wise error rate (the probability of committing a Type I error on *any* of the tests in the set) or the false discovery rate (the proportion of falsely rejected hypotheses among those rejected). Chapter 2 of Bretz et al. (2011) provides a brief, but informative discussion of these general concepts. While these are useful concepts, and the package discussed below permits users to adjust p-values in a number of ways to address these issues, I am more interested in how the multiple testing literature has developed around the presentation of multiple pairwise comparisons.

[5]The figure below subtracts the arbitrary constant from the results of `ftrend()` to put all of these estimates on the same scale. I recognize that this is not what the authors had intended, but this should not lead to erroneous inferences in any event (Easton and Peto, 2000).

[6]Horizontal gray lines have been drawn at the smallest lower- and largest upper-bounds of the mining sector floating confidence intervals to facilitate comparison. Note that differences across the three methods in the upper bounds and lower bounds were in the third decimal place.

Table 1: Analytical Test of Differences between Mining (MIN) and Other Sectors

Contrast	Estimate/(SE)
MIN - AGR	0.250*
	(0.069)
MIN - BNK	0.416*
	(0.084)
MIN - CON	0.739*
	(0.210)
MIN - FIN	0.361*
	(0.067)
MIN - HLD	0.265*
	(0.118)
MIN - MAN	0.128
	(0.071)
MIN - MER	0.188*
	(0.085)
MIN - TRN	0.098
	(0.071)
MIN - WOD	-0.248*
	(0.072)

* $p < 0.05$, two-sided.
Estimates and standard errors produced by `glht()` from the **multcomp** package.

Gramm et al. (2006) discuss the two generally accepted methods for presenting multiple comparisons — the line display and the letter display. A line display (see for example, Steel and Torrie, 1980) prints a column where each row represents a single element in the multiple comparisons. In the example above, using the Ornstein data, these would be the names of the various sectors. Then, vertical lines are drawn connecting all values that are not significantly different from each other. This is a relatively simple display, but as shown generally by Piepho (2004) and in this particular case, it is not always possible to faithfully represent all of the pairwise comparisons with connecting line segments. Note that in the third line, a discontinuity is required to properly depict all of the pairwise relationships. Further, this method requires that the levels of factors (at least potentially) be reordered to identify insignificant differences. This reordering, while reasonable for unordered factors, is not at all reasonable if the factor is inherently ordered. Figure 2(a) shows the line display for the Ornstein model above. A compact letter display (Piepho, 2004) places a series of letters by each level of the categorical variable such that any two levels with the same letter are not significantly different from each other.[7] Each letter essentially defines a set of factor levels that have insignificant differences in coefficients among them. For example, Banking, Construction and Finance all share the letter "a", which means their coefficients are statistically indistinguishable from each other. Note that Wood is the sole factor level with "f", meaning that it has a statistically different coefficient than all of the other factor levels. These are more flexible than line displays, though they can still be improved upon. Even though these displays do identify all pairwise significant relationships, they do not immediately identify the sign and size of the differences and what appear to be complicated patterns of significance may appear more simple with a different mode of display.

Graves et al. (2012) discuss enhancements to the letter display that make it somewhat more visually appealing and make the cognitive tasks involved less cumbersome. This method is operationalized by the `multcompTs` function in the **multcompView** package. While these functions are potentially useful, they are A) still improved upon by the method discussed below and B) not intended for use directly with "glm" class objects or "glht" class objects. Despite the improvements over letter displays, complicated patterns of (in)significance still result in cluttered displays.

[7]The boxplot on the graph is a boxplot of the linear predictor from the statistical model. If there were no other covariates in the model, this would just be a boxplot of the response variable by the different factor levels. While this does provide some information, it does not indicate how the predicted response changes as a function of the factor holding other things constant, which would perhaps be more useful.

Figure 1: Quasi-confidence Intervals for the Ornstein Model

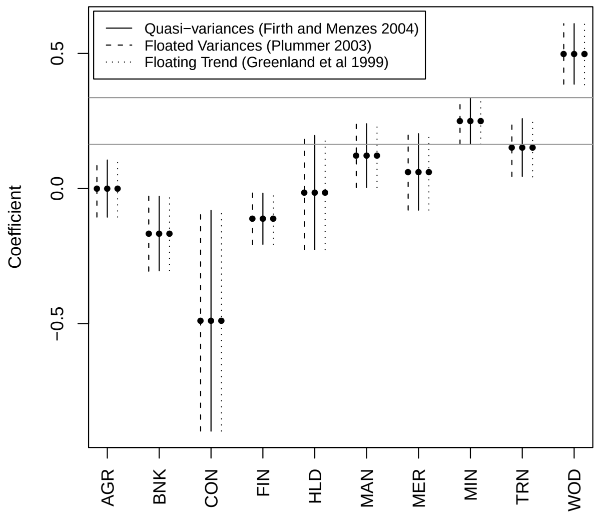
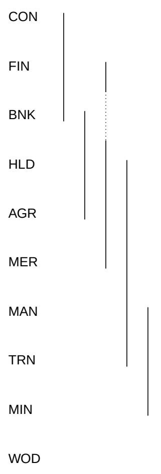
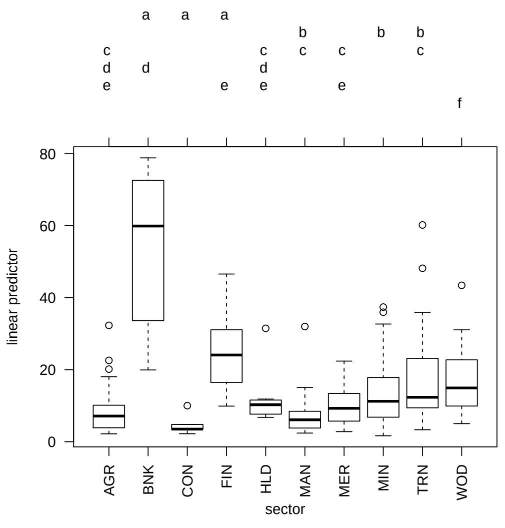


Figure 2: Line and Letter Displays for Ornstein Model

(a) Line Display



(b) Letter Display



An alternative method of presentation

I argue that a good solution to the reference category problem is one that permits the most efficient presentation and evaluation of a series of hypothesis tests relating to various (simple) factor contrasts. As discussed above, both the numerical presentation of floating variances and the visual presentation of floating confidence intervals are not maximally efficient on either dimension (presentation or evaluation) when the analyst desires information about the simple pairwise difference between coefficients related to the levels of a factor (i.e., simple contrasts). Similarly, I suggested that compact letter displays (and to a lesser extent multcompTs), though they present all of the appropriate information, are not maximally efficient at presenting the desired information graphically. As Chambers et al. (1983) and Cleveland (1985) suggest, one efficient way of presenting many pairwise relationships is through a scatterplot matrix or a generalized draftsman's display (a lower- or upper-triangular scatterplot matrix).[8] The important feature of a scatterplot matrix is the organization of pairwise displays in a common scale. Thus, a display that directly indicates the difference for the simple contrasts of interest would be superior to one that requires the user to make $(m(m-1))/2$ pairwise comparisons from m floating variances or confidence intervals.

The **factorplot** function in the package of the same name (version 1.1) for R computes all pairwise comparisons of coefficients relating to a factor; its print, summary and plot methods provide the user with a wealth of information regarding the nature of the differences in these coefficients.[9] These functions overcome the problems suffered by previous methods as they present the results of pairwise hypothesis tests directly in a visually appealing manner.

The function calculates equation (3) for each simple contrast directly through a set of elementary matrix operations. First, $\boldsymbol{d}$, a $m \times \frac{m(m-1)}{2}$ matrix in which each column has one entry equal to positive one, one entry equal to negative one and all the remaining entries equal to zero is created. The positive and negative ones indicate the comparison being calculated. Using the coefficients for the desired factor covariate (call them $\boldsymbol{g}$, a row-vector of length m), I calculate $\Delta = \boldsymbol{gd}$. Standard errors for contrasts are calculated using the m rows and columns of the variance-covariance matrix of the estimators from the model (call this $V(g)$): $V(\Delta) = \boldsymbol{d}'\boldsymbol{V}(g)\boldsymbol{d}$. The Δ vector and the square root of the diagonal of $V(\Delta)$ (both of length $\frac{m(m-1)}{2}$) are then organized into $(m-1) \times (m-1)$ upper-triangular matrices where the rows refer to the first $m-1$ elements of $\boldsymbol{g}$ and the columns refer to the last $m-1$ elements of $\boldsymbol{g}$. The entries indicate the difference between the coefficient represented by the row and the coefficient represented by the column and its standard error.

The function has methods for objects of class "lm", "glm", "glht" and "multinom" which do slightly different things depending on the input. The default method will accept a vector of estimates and either A) a full variance-covariance matrix or B) a vector of quasi or floated variances that will be turned into a diagonal variance-covariance matrix. The methods for "lm", "glm", "glht" and "summary.glht" objects calculate the pairwise differences in the linear predictor for the values of the specified factor variable. The method for "multinom" class objects calculates the pairwise differences in coefficients across the categories of the dependent variable for a single variable (i.e., column of the model matrix).

Example 1: Ornstein data

The `factorplot` method for "lm" class objects has six arguments. The first two arguments, `obj` and `adjust.method`, indicate the object and the method by which p-values are to be adjusted for multiple comparisons (possibilities include all of those to `p.adjust` from the **stats** package (R Core Team, 2013)). The `factor.variable` argument indicates the factor for which comparisons are desired. `pval` allows the user to set the desired Type I error rate and `two.sided` allows the user to specify whether the null hypothesis is tested against a one- or two-sided alternative with the latter as the default. The `order` argument sets the ordering of the coefficients, with three possibilities — 'natural', 'alph' and 'size'. The 'natural' option maintains the original ordering of the factor, the 'alph' option sorts them alphabetically and the 'size' option sorts in ascending order of the magnitude of the coefficient. The choices made here propagate through the plot, print and summary methods.

The plot method for "factorplot" class objects produces something akin to an upper-triangular scatterplot. The analogy is not perfect, but the idea is similar; each entry of the rows-by-columns display indicates the pairwise difference between coefficients. The statistical significance of these

[8] Cleveland (1985) makes the argument in favor of a full scatterplot matrix, but in this case, the information presented in the upper-triangle is sufficient as nothing new could be learned by examining the full square matrix.

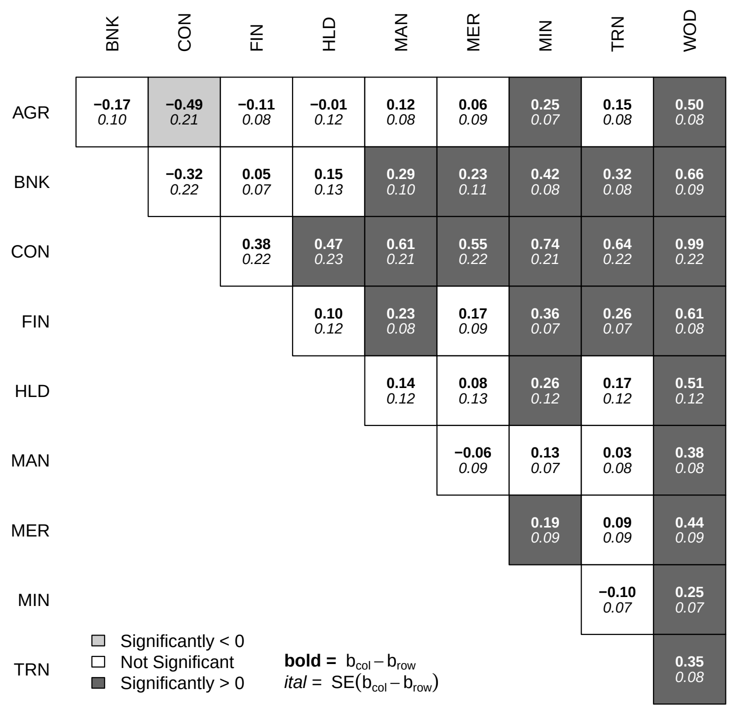
[9] The methods for "lm", "glm", "multinom" and the default method use the calculations mentioned below. The method for "glht" and "summary.glht" objects uses the built-in functionality from the **multcomp** package to do these calculations. The benefit here is that if a small subset of comparisons is desired, this subset can be identified in the call to `glht()` and only those comparisons will be computed, thus increasing efficiency.

differences is indicated by three colors (one for significant-positive, one for significant-negative and one for insignificant differences). The three colors can be controlled with the `polycol` argument and the text color within the polygons can be controlled with the `textcol` argument.[10] The plot method also allows the user to specify the number of characters with which to abbreviate the factor levels through the `abbrev.char` argument. Setting this to an arbitrarily high value will result in no abbreviation. Finally, the `trans` argument allows the user to impose a post-hypothesis-test transformation to the coefficient estimates. For example, if the underlying model is a logistic regression, tests will be done on the log-relative risks, but the relative risks could be plotted with `trans = "exp"`.[11] By default, the function prints legends identifying the colors and numbers; these can be turned on or off with the logical arguments `print.sig.leg` and `print.square.leg`, respectively. Figure 3 shows the display for the Ornstein model. The following code produces the result in the figure.

```
library(factorplot)
library(car)
mod <- glm(interlocks ~ log2(assets) + nation + sector, data = Ornstein,
    family = poisson)
fp <- factorplot(mod, adjust.method="none", factor.variable = "sector", pval = 0.05,
         two.sided = TRUE, order = "natural")
plot(fp, abbrev.char = 100)
```

The print method for a "factorplot" object prints all of the pairwise differences, their accompanying analytical standard errors and (optionally adjusted) p-values. The user can specify the desired number of decimal places for rounding, with the `digits` argument. The `sig` argument is logical allowing the user to print all pairwise differences if `FALSE` and only significant differences when `TRUE`. The print method also permits the same `trans` argument as the plot method for objects of class "factorplot". An example of the output from the print method is below. Here, twenty-five of the forty-five pairwise differences are statistically different from zero when.

Figure 3: Plotted factorplot object for Ornstein model



	BNK	CON	FIN	HLD	MAN	MER	MIN	TRN	WOD
AGR	□ **−0.17** *0.10*	▨ **−0.49** *0.21*	□ **−0.11** *0.08*	□ **−0.01** *0.12*	□ **0.12** *0.08*	□ **0.06** *0.09*	■ **0.25** *0.07*	□ **0.15** *0.08*	■ **0.50** *0.08*
BNK		□ **−0.32** *0.22*	□ **0.05** *0.07*	□ **0.15** *0.13*	■ **0.29** *0.10*	■ **0.23** *0.11*	■ **0.42** *0.08*	■ **0.32** *0.08*	■ **0.66** *0.09*
CON			□ **0.38** *0.22*	■ **0.47** *0.23*	■ **0.61** *0.21*	■ **0.55** *0.22*	■ **0.74** *0.21*	■ **0.64** *0.22*	■ **0.99** *0.22*
FIN				□ **0.10** *0.12*	■ **0.23** *0.08*	□ **0.17** *0.09*	■ **0.36** *0.07*	■ **0.26** *0.07*	■ **0.61** *0.08*
HLD					□ **0.14** *0.12*	□ **0.08** *0.13*	■ **0.26** *0.12*	□ **0.17** *0.12*	■ **0.51** *0.12*
MAN						□ **−0.06** *0.09*	□ **0.13** *0.07*	□ **0.03** *0.08*	■ **0.38** *0.08*
MER							■ **0.19** *0.09*	□ **0.09** *0.09*	■ **0.44** *0.09*
MIN								□ **−0.10** *0.07*	■ **0.25** *0.07*
TRN									■ **0.35** *0.08*

▨ Significantly < 0
□ Not Significant
■ Significantly > 0

bold = $b_{col} - b_{row}$
ital = $SE(b_{col} - b_{row})$

```
print(fp, sig = T)
          Difference    SE p.val
```

[10]The printing of the estimates and standard errors can both be turned off with `print.est = FALSE` and `print.se = FALSE`, respectively.

[11]After the hypothesis tests are done, a matrix named `r.bdiff` holds the coefficient differences. The transformation is done as follows: `do.call(trans, list(r.bdiff))`, so only transformations amenable to this procedure will work.

```
AGR - CON      0.489 0.213 0.023
CON - HLD     -0.474 0.235 0.045
BNK - MAN     -0.288 0.102 0.005
CON - MAN     -0.611 0.215 0.005
FIN - MAN     -0.233 0.082 0.005
BNK - MER     -0.228 0.106 0.032
CON - MER     -0.551 0.220 0.013
AGR - MIN     -0.250 0.069 0.000
BNK - MIN     -0.416 0.084 0.000
CON - MIN     -0.739 0.210 0.001
FIN - MIN     -0.361 0.067 0.000
HLD - MIN     -0.265 0.118 0.026
MER - MIN     -0.188 0.085 0.029
BNK - TRN     -0.318 0.082 0.000
CON - TRN     -0.641 0.217 0.004
FIN - TRN     -0.263 0.070 0.000
AGR - WOD     -0.498 0.076 0.000
BNK - WOD     -0.665 0.095 0.000
CON - WOD     -0.988 0.215 0.000
FIN - WOD     -0.610 0.077 0.000
HLD - WOD     -0.513 0.121 0.000
MAN - WOD     -0.376 0.080 0.000
MER - WOD     -0.437 0.090 0.000
MIN - WOD     -0.248 0.072 0.001
TRN - WOD     -0.346 0.081 0.000
```

The summary method for "factorplot" objects prints the number of coefficients that are significantly smaller than the one of interest and the number of coefficients larger than the one of interest for each level of the factor. While this is not a common means of presenting the results, this does nicely summarize the extent of significant differences among the coefficients. Below is an example of printout from the summary method. It is easy to see that the wood industry (WOD) has the highest conditional means as it is significantly bigger than all of other categories. It is also easy to see that the construction industry (CON) has one of the smallest conditional means as it is significantly smaller than seven of the other categories and not significantly bigger than any.

```
summary(fp)
    sig+ sig- insig
AGR    1    2     6
BNK    0    5     4
CON    0    7     2
FIN    0    4     5
HLD    1    2     6
MAN    3    1     5
MER    2    2     5
MIN    6    1     2
TRN    3    1     5
WOD    9    0     0
```

Together, the `factorplot` function and its associated print, plot and summary methods provide a wealth of information including direct hypothesis tests using analytical standard errors for the simple contrasts most commonly desired in (G)LMs.

Example 2: *H. pylori* and gastric precancerous lesions

Plummer et al. (2007) were interested in discerning the extent to which infection with *H. pylori* containing the cytotoxin-associated (cagA) gene increased the severity of gastric precancerous lesions. They found that cagA+ patients had increased risks of more severe lesions while cagA- patients were only at significantly higher risk (than their uninfected counterparts) of chronic gastritis. Table 2 summarizes the results of the relative risk of the various types of gastric lesions versus the baseline of normal or superficial gastritis.

The default method for the `factorplot` function allows the user to supply a vector of point estimates and (floating) variances rather than an estimated model object. This function will be particularly useful for those scholars in epidemiology, where floating standard errors are more routinely presented. With 7 levels of the factor in Table 2, there are 21 pairwise comparisons implied, which would require

Table 2: Results from Plummer et al. (2007)

	cagA-		cagA+	
	OR	FSE	OR	FSE
Normal and superficial gastritis	1.00	0.242	1.00	0.320
Chronic gastritis	2.12	0.096	4.33	0.101
Chronic atrophic gastritis	1.44	0.156	3.89	0.160
Intestinal metaplasia I	1.31	0.140	4.14	0.141
Intestinal metaplasia II	1.44	0.380	10.8	0.349
Intestinal metaplasia III	1.46	0.484	21.9	0.431
Dysplasia	0.90	0.375	15.5	0.311

OR = odds ratio
FSE = floating standard error
Adapted from Figure 1 in Plummer et al. (2007, p1331).

users to do a lot of calculations. However, inputting the estimates and floated variances into R and subjecting them to the `factorplot` function can do all of the calculations automatically. Below is an example of how the results could be used in conjunction with the **factorplot** suite of functions.

```
est1 <- log(c(1.00,2.12,1.44,1.31,1.44,1.46,0.90))
var1 <- c(0.242,0.096,0.156,0.140,0.380,0.484,0.375)^2
est2 <- log(c(1.00,4.33,3.89,4.14,10.8,21.9,15.5))
var2 <- c(0.320,0.101,0.160,0.141,0.349,0.431,0.311)^2
resdf <- 48+16+27+532+346+144+144+124+58+166+162+75+24+
        53+10+15+61+6+18+90+12-18
names(est1) <- names(est2) <- c(
     "Normal Gas","Chronic Gas", "Chronic A. Gas",
     "IM I", "IM II", "IM III", "Dysplasia")

plummer_fp1 <- factorplot(est1, var = var1, resdf = resdf, adjust.method = "none")
plummer_fp2 <- factorplot(est2, var = var2, resdf = resdf, adjust.method = "none")
plot(plummer_fp1, trans = "exp", abbrev.char = 100, scale.text = 1.5,
    scale.space = 1.5)
plot(plummer_fp2, trans = "exp", abbrev.char = 100, scale.text = 1.5,
    scale.space = 1.5)
```

The plots are displayed in Figure 4. The left-hand plot suggests that *H. pylori* cagA- seems to raise the risk of chronic gastritis relative to Intestinal metaplasia I and the reference group of normal and superficial gastritis. The differences in the risk of chronic gastritis and chronic atrophic gastritis or dysplasia are also significant. The right-hand plot indicates that there are no significant differences among the second through fourth diagnoses and the fifth through seventh diagnoses. The difference between the risk of intestinal metaplasia I and II (for cagA+) is also significant.

Example 3: vote choice in France

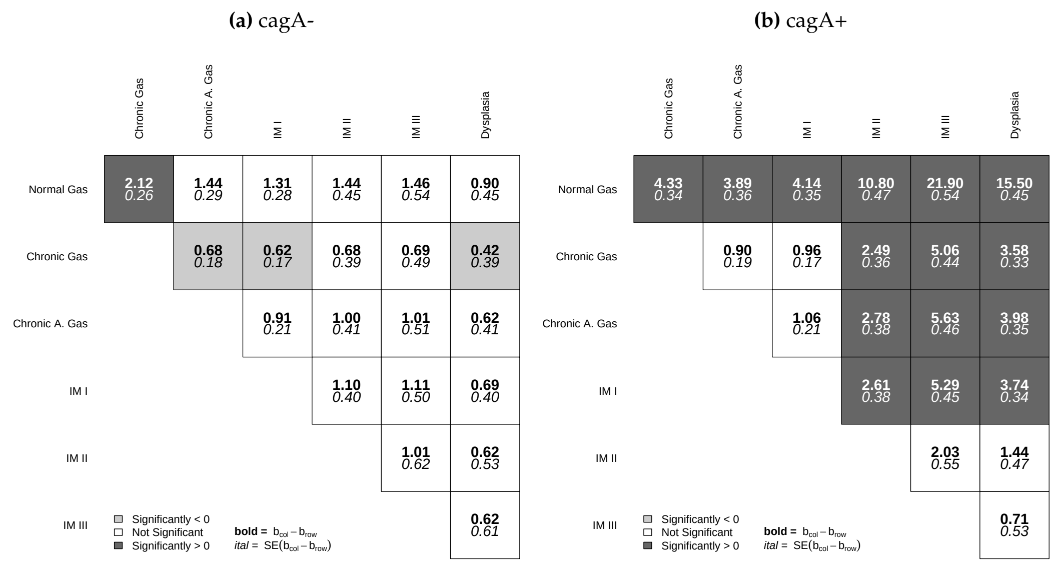
When `factorplot()` encounters an object of class `multinom`, it will make comparisons within the same variable across all levels of the dependent variable. The coefficient table presents a specific set of pairwise comparisons — namely those indicating the relationship of each variable to the binary choice of each non-reference category versus the reference category. However, other comparisons implied by that coefficient table may be interesting or useful and should be investigated.

In the example below, I estimate a multinomial logistic regression model of vote choice (`vote`) on a number of standard controls: retrospective national economic evaluations (`retnat`), self-placement on the left-right ideological continuum (`lrself`), gender (`male`) and age (`age`).[12]

```
library(nnet)
data(france)
france.mod <- multinom(vote ~ retnat + lrself + male + age, data = france)
```

[12]See help for `france` in the package **factorplot** for more details about the origin and coding of the data.

Figure 4: Results from Plummer et al. (2007) Presented as factorplots



```
fp3 <- factorplot(france.mod, variable = "age")
plot(fp3)
```

Figure 5 shows that as people get older, they are more likely to vote for RPR or UDF than the Greens or Communists (PCF) and more likely to vote for the Socialists (PS) than the Greens. If one is interested in whether variables have significant effects on vote choice, all pairwise comparisons should be considered. `factorplot` makes it easy for users to appropriately evaluate all relevant pairwise comparisons.

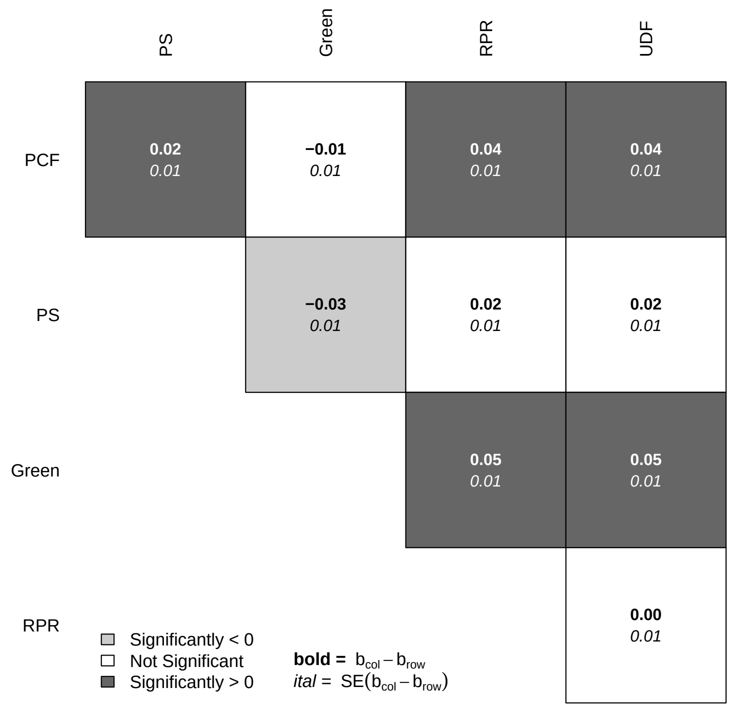
Conclusion

Easton's (1991) contribution of floating absolute risk has been influential, especially in epidemiology and medicine, allowing researchers to present easily information that permits the reader to make any pairwise comparison among the different levels of a risk factor. Firth and De Menezes (2004); de Menezes (1999) and Plummer (2004) have provided not only a rigorous, model-based foundation for this idea, but have also provided software that easily produces these quantities for a wide array of statistical models. I argue that while these quantities are interesting and useful, floating confidence intervals, which are often provided ostensibly to permit hypothesis testing can be imprecise and potentially misleading, as regards hypothesis testing. Compact letter displays (Piepho, 2004) are a step in the right direction, but I argue that they can still be improved upon in terms of graphically presenting information of interest to many researchers. In the common situation wherein one is interested in simple contrasts, the `factorplot()` functions and their associated print, plot and summary methods discussed above provide much greater transparency with respect to the presentation and evaluation of hypothesis tests than floating absolute risk or quasi-variance estimates. The visual presentation of direct hypothesis tests requires much less effort to adjudicate significance and uncover patterns in the results than other methods, including compact letter displays. While the calculation of these hypothesis tests is not novel, the methods of presenting and summarizing the information represent a significant advance over the previously available general solutions available in R.

Acknowledgements

I would like to thank Bob Andersen, Ryan Bakker, John Fox, Bill Jacoby, Martyn Plummer and an anonymous reviewer as well as participants in the Regression III course at the ICPSR Summer Program for helpful comments and suggestions.

Figure 5: Plotted factorplot object for Age from Multinomial Logit model



Bibliography

Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society Series B*, 57(1):289–300, 1995. [p5]

F. Bretz, T. Hothorn, and P. Westfall. *Multiple Comparisons Using R*. Taylor and Francis Group, LLC, Boca Raton, FL, 2011. [p6]

R. H. Browne. Visual assessment of the significance of a mean difference. *Biometrics*, 35(3):657–665, 1979. [p6]

B. Carstensen, M. Plummer, E. Laara, and M. Hills. *Epi: A Package for Statistical Analysis in Epidemiology*, 2013. URL `http://CRAN.R-project.org/package=Epi`. R package version 1.1.44. [p5]

J. M. Chambers, W. S. Cleveland, B. Kleiner, and P. A. Tukey. *Graphical Methods for Data Analysis*. Wadsworth & Brooks/Cole Publishing Company, Pacific Grove, CA, 1983. [p9]

W. Cleveland. *Elements of Graphing Data*. Wadsworth, Inc., Monterey, CA, 1985. [p5, 6, 9]

R. X. de Menezes. *More Useful Standard Errors for Group and Factor Effects in Generalized Linear Models*. PhD thesis, Department of Statistics, University of Oxford, 1999. [p4, 13]

D. F. Easton and J. Peto. Presenting statistical uncertainty in trends and dose-response relationships (letter). *American Journal of Epidemiology*, 152(393), 2000. [p5, 6]

D. F. Easton, J. Peto, and A. Babiker. Floating absolute risk: An alternative to relative risk in survival and case-control analysis avoiding an arbitrary reference group. *Statistics in Medicine*, 10:1025–1035, 1991. [p4, 5]

D. Firth. *qvcalc: Quasi variances for factor effects in statistical models*, 2010. URL `http://CRAN.R-project.org/package=qvcalc`. [p5]

D. Firth and R. X. De Menezes. Quasi-variances. *Biometrika*, 91(1):65–80, 2004. [p4, 5, 6, 13]

J. Fox and S. Weisberg. *An R Companion to Applied Regression*. Sage, Thousand Oaks CA, second edition, 2011. [p6]

J. Gramm, J. Guo, F. Hüffner, R. N. B, H. peter Piepho, and R. Schmid. Algorithms for compact letter displays: Comparison and evaluation. *Computational Statistics & Data Analysis*, 52(12):725–736, 2006. [p5, 6]

S. Graves, H.-P. Piepho, L. Selzer, and S. Dorai-Raj. *multcompView: Visualizations of Paired Comparisons*, 2012. URL `http://CRAN.R-project.org/package=multcompView`. R package version 0.1-5. [p5, 7]

S. Greenland, K. B. Michels, and J. M. Robins. Presenting statistical uncertainty in trends and dose-response relations. *American journal of . . .*, 149(12):1077–1086, 1999. [p5]

S. Greenland, K. Michels, C. Poole, and W. Willett. Four of the authors reply [re: "presenting statistical uncertainty in trends and dose-response relationships"] (letter). *American Journal of Epidemiology*, 152:394, 2000. [p5]

S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6: 65–70, 1979. [p5]

T. Hothorn, F. Bretz, and P. Westfall. Simultaneous inference in general parametric models. *Biometrical Journal*, 50:346–363, 2008. [p4]

M. Ornstein. The boards and executives of the largest canadian corporations. *Canadian Journal of Sociology*, 1:411–437, 1976. [p6]

H. Piepho. An algorithm for a letter-based representation of all pairwise comparisons. *Journal of Computational and Graphical Statistics*, 13:456–466, 2004. [p7, 13]

M. Plummer. Improved estimates of floating absolute risk. *Statistics in Medicine*, 23:93–104, 2004. [p4, 5, 6, 13]

M. Plummer, L. van Doorn, S. Franceschi, B. Kleter, F. Canzian, J. Vivas, G. Lopez, D. Colin, N. Muñoz, and I. Kato. Helicobacter pylori cytotoxin-associated genotype and gastric precancerous lesions. *Journal of the National Cancer Institute*, 99:1328–1334, 2007. [p11, 12]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013. URL `http://www.R-project.org/`. [p9]

R. W. Smith. Visual hypothesis testing with confidence intervals. *Proceedings of the 22^{n}d SAS User's Group*, pages 1252–1257, 1997. [p6]

R. Steel and J. Torrie. *Principles and Procedures of Statistics*. McGraw-Hill, New York, 1980. [p5, 7]

David A. Armstrong II
University of Wisconsin - Milwaukee
Department of Political Science
P.O. Box 413
Milwaukee, WI 53201
United States of America
`armstrod@uwm.edu`

spMC: Modelling Spatial Random Fields with Continuous Lag Markov Chains

by Luca Sartore

Abstract Currently, a part of the R statistical software is developed in order to deal with spatial models. More specifically, some available packages allow the user to analyse categorical spatial random patterns. However, only the **spMC** package considers a viewpoint based on transition probabilities between locations. Through the use of this package it is possible to analyse the spatial variability of data, make inference, predict and simulate the categorical classes in unobserved sites. An example is presented by analysing the well-known Swiss Jura data set.

Introduction

Originally, the **spMC** package (Sartore, 2013) was developed with the purpose of analysing categorical data observed in 3-D locations. It deals with stochastic models based on Markov chains, which may be used for the analysis of spatial random patterns in continuous multidimensional spaces (Carle and Fogg, 1997). Its results are easily interpretable and it is a good alternative to the T-PROGS software developed by Carle (1999), which is oriented towards modelling groundwater systems. The models considered in the **spMC** package are used to analyse any categorical random variable $Z(\mathbf{s})$ at the d-dimensional position $\mathbf{s} \in \mathbb{R}^d$ which satisfies the Markov property. Other R packages are also helpful for analysing categorical spatial data. For example, the **gstat** package (Pebesma, 2004) allows for analyses using traditional methods such as the parameter estimation of spatial models based on variograms and kriging techniques for predictions. All these methods and their variants are also available in other packages, e.g. **geoRglm** (Christensen and Ribeiro Jr, 2002) and **RandomFields** (Schlather, 2013). When $Z(\mathbf{s})$ is assumed to be linked to a continuous hidden random process, these packages are useful for studying the covariance structure of the data.

The **spMC** package extends the functionality of the T-PROGS software to R users. New useful functions are included for faster modelling of transition probability matrices, and efficient algorithms are implemented for improving predictions and simulations of categorical random fields. The main tasks and their functions are clearly summarised in Table 1. Three different fitting methods were implemented in the package. The first is based on the estimates of the main features that characterise the process, the second focuses on the minimisation of the discrepancies between the empirical and theoretical transition probabilities, and the third follows the maximum entropy approach. Once the model parameters are properly estimated, transition probabilities are calculated through the matrix-valued exponential function (see Higham, 2008, Algorithm 10.20 in Chapter 10). These transition probabilities are then combined to predict the category in an unsampled position. Three algorithms are used to simulate spatial random fields; those based on the kriging techniques (Carle and Fogg, 1996), those using fixed and random path methods (Li, 2007a; Li and Zhang, 2007), or those using multinomial categorical simulation proposed by Allard et al. (2011). In order to reduce computation time through OpenMP API (version 3.0; OpenMP Architecture Review Board, 2008), the `setCores()` function allows the user to change the number of CPU cores, so that one can mix shared memory parallel techniques with those based on the Message Passing Interface (The MPI Forum, 1993) as described in Smith (2000).

Here, it will be shown how to perform a geostatistical analysis of the Jura data set (Goovaerts, 1997) using the **spMC** package (version 0.3.1). The data set consists of 359 sampled spatial coordinates and their respective observed realisations of two categorical variables (related to the rock-type and the land use) and some continuous variables (corresponding to the topsoil content).

Brief overview of the models

The **spMC** package deals with both one-dimensional and multidimensional continuous lag models. If $Z(\mathbf{s}_l)$ denotes a categorical random variable in a location $\mathbf{s}_l$, for any $l = 1, \ldots, n$, its outcome conventionally takes values in the set of mutually exclusive states $\{z_1, \ldots, z_K\}$, where K represents the total number of observable categories. A continuous lag Markov chain model organises the conditional probabilities

$$t_{ij}(\mathbf{s}_l - \mathbf{s}_k) = \Pr(Z(\mathbf{s}_l) = z_j | Z(\mathbf{s}_k) = z_i),$$

for any $i, j = 1, \ldots, K$, in a $K \times K$ transition probability matrix. Generally speaking, such a model is a transition probability matrix-valued function depending on one-dimensional or multidimensional

Tasks and functions	Techniques implemented in the **spMC** package
Estimations of one-dimensional continuous lag models	
`transiogram`	Empirical transition probabilities estimation
`tpfit`	One-dimensional model parameters estimation
`tpfit_ils`	Iterated least squares method for one-dimensional model parameters estimation
`tpfit_me`	Maximum entropy method for one-dimensional model parameters estimation
`tpfit_ml`	Mean length method for one-dimensional model parameters estimation
Estimations of multidimensional continuous lag models	
`pemt`	Pseudo-empirical multidimensional transiograms estimation
`multi_tpfit`	Multidimensional model parameters estimation
`multi_tpfit_ils`	Iterated least squares method for multidimensional model parameters estimation
`multi_tpfit_me`	Maximum entropy method for multidimensional model parameters estimation
`multi_tpfit_ml`	Mean length method for multidimensional model parameters estimation
Categorical spatial random field simulation and prediction	
`sim`	Random field simulation
`sim_ck`	Conditional simulation based on indicator cokriging
`sim_ik`	Conditional simulation based on indicator kriging
`sim_mcs`	Multinomial categorical simulation
`sim_path`	Conditional simulation based on path algorithms
Graphical tools	
`plot.transiogram`	Plot one-dimensional transiograms
`mixplot`	Plot of multiple one-dimensional transiograms
`contour.pemt`	Display contours with pseudo-empirical multidimensional transiograms
`image.pemt`	Images with pseudo-empirical multidimensional transiograms
`image.multi_tpfit`	Images with multidimensional transiograms

Table 1: Most important user functions in the **spMC** package.

lags, i.e.

$$\mathbf{T}(h_\phi) : \mathbb{R}^d \to [0,1]^{K\times K},$$

wherein h_ϕ denotes a d-dimensional continuous lag along the direction $\phi \in \mathbb{R}^d$. Such a lag corresponds to the difference between the location coordinates and is proportional to the direction ϕ. The exponential form,

$$\begin{aligned}\mathbf{T}(h_\phi) &= \exp\left(\|h_\phi\| \mathbf{R}_\phi\right), \\ &= \sum_{u=0}^{\infty} \frac{\|h_\phi\|^u}{u!} \mathbf{R}_\phi^u,\end{aligned} \tag{1}$$

is usually adopted to model the observed variability and local anisotropy. The components of the transition rate matrix $\mathbf{R}_\phi \in \mathbb{R}^{K\times K}$ (the model coefficients) depend on the direction ϕ and they must satisfy the following properties (Norris, 1998, Section 2.1):

- $r_{ii} \leq 0$, for any $i = 1, \dots, K$.
- $r_{ij} \geq 0$, if $i \neq j$.
- The row sums satisfy
$$\sum_{j=1}^{K} r_{ij} = 0.$$
- The column sums satisfy
$$\sum_{i=1}^{K} p_i r_{ij} = 0,$$
where p_i is the proportion of the i-th category.

The components of $\mathbf{R}_{-\phi}$ may be computed through the relation

$$r_{ij,\,-\phi} = \frac{p_j}{p_i} r_{ji,\,\phi} \qquad \forall i, j = 1, \ldots, K,$$

where $-\phi$ denotes the opposite direction.

Transition rate matrix estimation

In order to obtain an estimate of the transition rate matrix along the direction ϕ, the package provides two solutions, i.e. by following the one-dimensional approach or the multidimensional. The latter estimates the matrix $\mathbf{R}_\phi$ by the ellipsoidal interpolation of d matrices, which are computed along the axial directions through one-dimensional procedures.

The one-dimensional techniques related to the `tpfit_ml()` and `tpfit_me()` functions are based on mean lengths $\bar{L}_{i,\,\phi}$ and transition frequencies of embedded occurrences $f^*_{kj,\,\phi}$. The iterated least squares method is implemented through the `tpfit_ils()` function.

The first two functions estimate the stratum mean lengths for each category through the `mlen()` function. The mean lengths are computed either with the average of the observed stratum lengths or their expectation based on the maximum likelihood estimate by assuming that the observed lengths are independent realisations of a log-normal random variable. In order to verify the distributional assumption on the lengths, the function `getlen()` estimates stratum lengths of embedded Markov chains along a chosen direction, while other functions such as `boxplot.lengths()`, `density.lengths()`, `hist.lengths()` are used for graphical diagnostics.

The `tpfit_ml()` function computes the transition frequencies of embedded occurrences as an average through the function `embed_MC()`. The maximum entropy method, adopted by the `tpfit_me()` function, calculates the transition frequencies of embedded occurrences through the iterative proportion fitting (Goodman, 1968). The algorithm may be summarised as follows:

1. Initialise $f_{i,\,\phi}$ with $p_i / \bar{L}_{i,\,\phi}$.
2. Compute $f^*_{ij,\,\phi} = f_{i,\,\phi} f_{j,\,\phi}$ $\forall i, j = 1, \ldots, K$.
3. Compute
$$f_{i,\,\phi} = \frac{p_i \sum_{k=1}^K \sum_{j \neq k}^K f^*_{kj,\,\phi}}{\bar{L}_{i,\,\phi} \sum_{j \neq i}^K f^*_{ij,\,\phi}}.$$
4. Repeat the second and the third step until convergence.

Both `tpfit_ml()` and `tpfit_me()` functions estimate the autotransition rates as $r_{ii} = -1/\bar{L}_{i,\,\phi}$, while the rates for any $i \neq j$ are calculated as $r_{ij,\,\phi} = f^*_{ij,\,\phi} / \bar{L}_{i,\,\phi}$.

The `tpfit_ils()` function estimates the transition rate matrix by minimising the sum of the squared discrepancies between the empirical probabilities given by the `transiogram()` function and theoretical probabilities given by the model. The bound-constrained Lagrangian method (Conn et al., 1991) is performed in order to have a proper transition rate matrix, which satisfies the transition rate properties.

The multidimensional approach is computationally efficient. In fact a generic entry of the matrix $\mathbf{R}_\phi$ is calculated by the ellipsoidal interpolation as

$$|r_{ij,\,\phi}| = \sqrt{\sum_{v=1}^{d} \left(\frac{h_{v,\,\phi}}{\|h_\phi\|} r_{ij,\,\mathbf{e}_v} \right)^2}, \tag{2}$$

where $h_{v,\,\phi}$ is the v-th component of the vector h_ϕ, $\mathbf{e}_v$ represents the standard basis vector, and the rate $r_{ij,\,\mathbf{e}_v}$ is replaced by $r_{ij,\,-\mathbf{e}_v}$ for components $h_{v,\,\phi} < 0$. In this way, it is only necessary to have in memory the estimates for the main directions.

The `multi_tpfit_ml()`, `multi_tpfit_me()` and the `multi_tpfit_ils()` functions automatically perform the estimation of d transition rate matrices along the axial directions with respect to the chosen one-dimensional method.

Prediction and simulation based on transition probabilities

Several methods were developed to predict or simulate the category in an unobserved location $\mathbf{s}_0$ given the knowledge of the sample positions $\mathbf{s}_1, \ldots, \mathbf{s}_n$. The conditional probability,

$$\Pr\left(Z(\mathbf{s}_0) = z_i \left| \bigcap_{l=1}^{n} Z(\mathbf{s}_l) = z(\mathbf{s}_l) \right. \right),$$

is used to predict or simulate the category in $\mathbf{s}_0$, where z_i represents the i-th category and $z(\mathbf{s}_l)$ is the observed category in the l-th sample location.

Usually such a probability is approximated through

- Kriging- and Cokriging-based methods, implemented in the `sim_ik()` and `sim_ck()` functions.
- Fixed or random path algorithms, available in `sim_path()`.
- Multinomial categorical simulation procedure, `sim_mcs()` function.

The approximation proposed by Carle and Fogg (1996) is implemented in the functions `sim_ik()` and `sim_ck()`. Both of them use some variant of the following

$$\Pr\left(Z(\mathbf{s}_0) = z_j \left| \bigcap_{l=1}^{n} Z(\mathbf{s}_l) = z(\mathbf{s}_l) \right. \right) \approx \sum_{l=1}^{n} \sum_{i=1}^{K} w_{ij,\,l}\, c_{il},$$

where

$$c_{il} = \begin{cases} 1 & \text{if } z(\mathbf{s}_l) = z_i, \\ 0 & \text{otherwise,} \end{cases}$$

and the weights $w_{ij,\,l}$ are calculated by solving the following system of linear equations:

$$\begin{bmatrix} \mathbf{T}(\mathbf{s}_1 - \mathbf{s}_1) & \cdots & \mathbf{T}(\mathbf{s}_n - \mathbf{s}_1) \\ \vdots & \ddots & \vdots \\ \mathbf{T}(\mathbf{s}_1 - \mathbf{s}_n) & \cdots & \mathbf{T}(\mathbf{s}_n - \mathbf{s}_n) \end{bmatrix} \begin{bmatrix} \mathbf{W}_1 \\ \vdots \\ \mathbf{W}_n \end{bmatrix} = \begin{bmatrix} \mathbf{T}(\mathbf{s}_0 - \mathbf{s}_1) \\ \vdots \\ \mathbf{T}(\mathbf{s}_0 - \mathbf{s}_n) \end{bmatrix},$$

where

$$\mathbf{W}_l = \begin{bmatrix} w_{11,\,l} & \cdots & w_{1K,\,l} \\ \vdots & \ddots & \vdots \\ w_{K1,\,l} & \cdots & w_{KK,\,l} \end{bmatrix}.$$

This approximation does not satisfy the probability axioms, because such probabilities might lie outside the interval $[0, 1]$ and it is not ensured that they sum up to one. To solve the former problem truncation is considered, but the usual normalisation is not adopted to solve the latter; in fact, after the truncation, these probabilities might also sum up to zero instead of one. The implemented stabilisation algorithm translates the probabilities with respect to the minimum computed for that point. Then, the probabilities are normalised as usual.

To improve the computational efficiency of the algorithm, the m-nearest neighbours are considered in the system of equations instead of all sample points; in so doing, a decrease in computing time is noted and the allocated memory is drastically reduced to a feasible quantity.

For the approximation adopted in the `sim_path()` function, conditional independence is assumed in order to approximate the conditional probability as in the analysis of a Pickard random field (Pickard, 1980). This method, as described in Li (2007b), considers m known neighbours in the axial directions, so that the probability is computed as

$$\Pr\left(Z(\mathbf{s}_0) = z_i \left| \bigcap_{l=1}^{n} Z(\mathbf{s}_l) = z(\mathbf{s}_l) \right. \right) \approx \Pr\left(Z(\mathbf{s}_0) = z_i \left| \bigcap_{l=1}^{m} Z(\mathbf{s}_l) = z_{k_l} \right. \right) \propto$$
$$\propto t_{k_1 i}(\mathbf{s}_0 - \mathbf{s}_1) \prod_{l=2}^{m} t_{ik_l}(\mathbf{s}_0 - \mathbf{s}_l).$$

The method proposed by Allard et al. (2011) is implemented in the `sim_mcs()` function. It was introduced to improve the computational efficiency of the Bayesian maximum entropy approach proposed by Bogaert (2002). Here, the approximation of the conditional probability is

$$\Pr\left(Z(\mathbf{s}_0) = z_i \,\middle|\, \bigcap_{l=1}^{n} Z(\mathbf{s}_l) = z(\mathbf{s}_l)\right) \approx \frac{p_i \prod_{l=1}^{n} t_{ik_l}(\mathbf{s}_0 - \mathbf{s}_l)}{\sum_{i=1}^{K} p_i \prod_{l=1}^{n} t_{ik_l}(\mathbf{s}_0 - \mathbf{s}_l)}.$$

Also in this case, the user can choose to apply this approximation by considering all data or only the m-nearest neighbours, with the same advantages described above.

Once the conditional probabilities are computed, the prediction is given by the highest probable category, while the simulation is given by randomly selecting one category according to the computed probabilities.

After the first simulation is drawn, the `sim_ik()` and `sim_ck()` functions execute an optimisation phase in order to avoid "artifact discontinuities", wherein the simulated patterns do not collimate with the conditioning data (Carle, 1997). The user can then choose to perform simulated annealing or apply a genetic algorithm in order to reduce the quantity

$$\sum_{i=1}^{K}\sum_{j=1}^{K}\left(r_{ij,\,\mathrm{SIM}} - r_{ij,\,\mathrm{MOD}}\right)^2 + \sum_{i=1}^{K}\left(p_{i,\,\mathrm{SIM}} - p_{i,\,\mathrm{MOD}}\right)^2,$$

where $r_{ij,\,\mathrm{SIM}}$ and $p_{i,\,\mathrm{SIM}}$ are coefficients estimated from the pattern to optimise, while $r_{ij,\,\mathrm{MOD}}$ and $p_{i,\,\mathrm{MOD}}$ are those used to generate the initial simulation. Other comparison methods are also available through the argument `optype`.

An example with the Jura data set

The data set consists of spatial coordinates and the observed values of both categorical and continuous random variables collected at 359 locations in the Jura region in Switzerland. In particular, we will deal with the rock-type categorical variable of the geological map created by Goovaerts (1997, see Figure 4), which consists of 5957 sites. The aim of these analyses is related to the parameters estimation of the model in (1), and its interpretation through graphical methods. These analyses are useful to check the model assumptions and to ensure the accuracy of the predictions.

First, the **spMC** package and the Jura data set in the **gstat** package are loaded as follows:

```
library(spMC)
data(jura, package = "gstat")
```

If the package is compiled with the OpenMP API, the number of CPU cores to use can be set by

```
setCores(4)
```

otherwise a message will be displayed and only one core can be internally used by the **spMC** package.

In order to study the spatial variability of the data and interpret the transitions from a geometrical viewpoint, the empirical transition probabilities along the main axes are calculated. These probabilities point out the persistence of a category according to the lag between two points. They also provide juxtapositional and asymmetrical features of the process, which are not detected by adopting indicator cross-variograms (Carle and Fogg, 1996). Therefore, all couples of points along axial directions are chosen such that their lag-length is less than three. After, we calculate the empirical transition probabilities for twenty points within the maximum distance. This can be conducted with the execution of the following code:

```
data <- jura.grid[, 4]
coords <- jura.grid[, 1:2]
Trg <- list()
Trg[[1]] <- transiogram(data, coords, max.dist = 3, mpoints = 20,
                        direction = c(1, 0))
Trg[[2]] <- transiogram(data, coords, max.dist = 3, mpoints = 20,
                        direction = c(0, 1))
```

If we want to compare these probabilities with the theoretical one, we first need to estimate two transition rate matrices, i.e. the model coefficients, along the axial directions. Three estimation methods are available in the **spMC** package (see Table 1), but only those based on mean lengths and maximum entropy are shown, even though the iterated least squares may be similarly applied. The code to estimate the transition rates through the mean lengths method is written as follows:

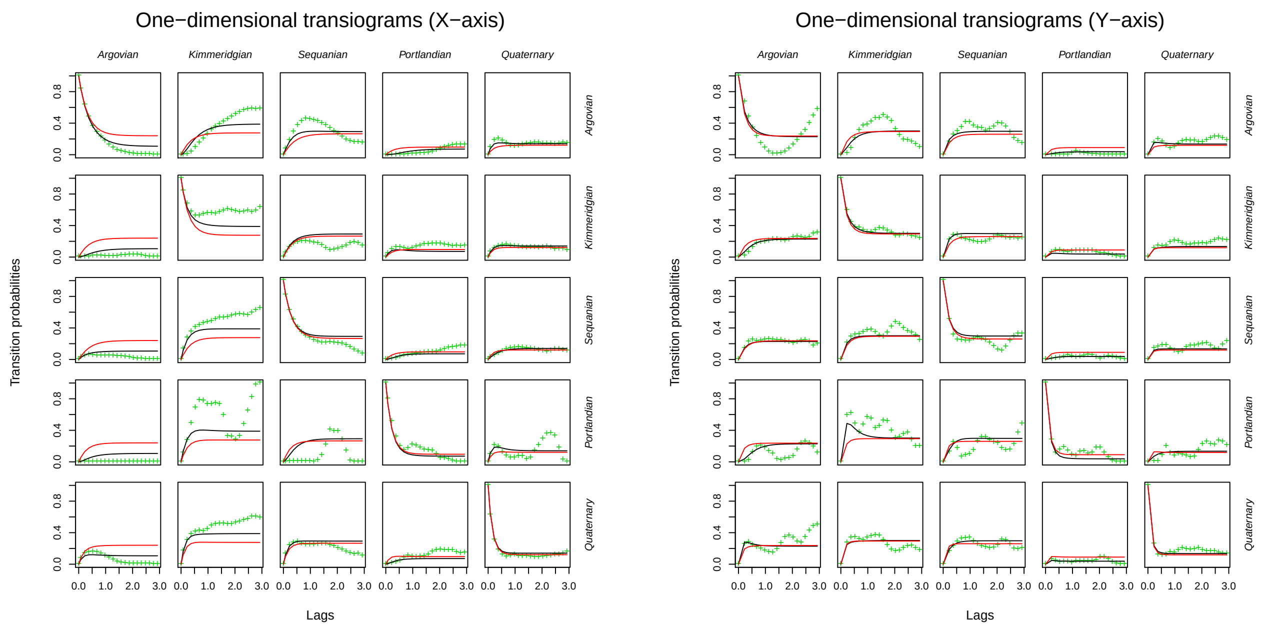


Figure 1: Empirical transiogram (green points) and theoretical probabilities (average method in black lines, maximum entropy method in red) along the X-axis (left) and along the Y-axis (right).

```
RTm <- list()
RTm[[1]] <- tpfit_ml(data, coords, direction = c(1, 0))
RTm[[2]] <- tpfit_ml(data, coords, direction = c(0, 1))
```

In this case, the mean lengths are calculated through the average of the stratum lengths along the chosen directions. On the other hand, to estimate the transition rate matrices through the maximum entropy approach, the following code must be executed:

```
ETm <- list()
ETm[[1]] <- tpfit_me(data, coords, direction = c(1, 0))
ETm[[2]] <- tpfit_me(data, coords, direction = c(0, 1))
```

Given the model coefficients, the transition probabilities for some specific lags are calculated as in (1). This is done as follows:

```
RTr <- list()
ETr <- list()
for (i in 1:2) {
    RTr[[i]] <- predict(RTm[[i]], lags = Trg[[i]]$lags)
    ETr[[i]] <- predict(ETm[[i]], lags = Trg[[i]]$lags)
}
```

Since these probabilities are calculated with respect to some fixed directions, i.e. by considering a one-dimensional perspective, they can be graphically compared. By the use of the `mixplot()` function, several transition probability diagrams (*transiograms*) can be superposed in a unique graphic, e.g.

```
for (i in 1:2)
  mixplot(list(Trg[[i]], RTr[[i]], ETr[[i]]), type = c("p", "l", "l"), pch = "+",
          col = c(3, 1, 2), legend = FALSE, main = paste(
          "One-dimensional transiograms", c("(X-axis)", "(Y-axis)")[i]))
```

By looking at the graphics in Figure 1, one can see how well the estimated models fit the observed probabilities (green points). This kind of graphic may be interpreted as a transition probability matrix; in fact it shows the probability dynamic related to one-dimensional lags along the specified direction. For example, let us consider the first horizontal line of graphics in Figure 1 on the left. They denote the transition probabilities at each lag from the Argovian state to one of the five categories.

These graphics are mainly used to investigate the stationarity of the stochastic process. In particular, the process is weakly stationary if the expected probabilities are not dependent on the location points, i.e. $\mathbb{E}[\Pr(Z(\mathbf{s}) = z_i)] = p_i$ for all $\mathbf{s} \in \mathbb{R}^d$. Theoretically, the transition probability matrix in (1) becomes constant as the lag distance $\|h_\phi\| \to \infty$. In order to check the stationarity property of the process

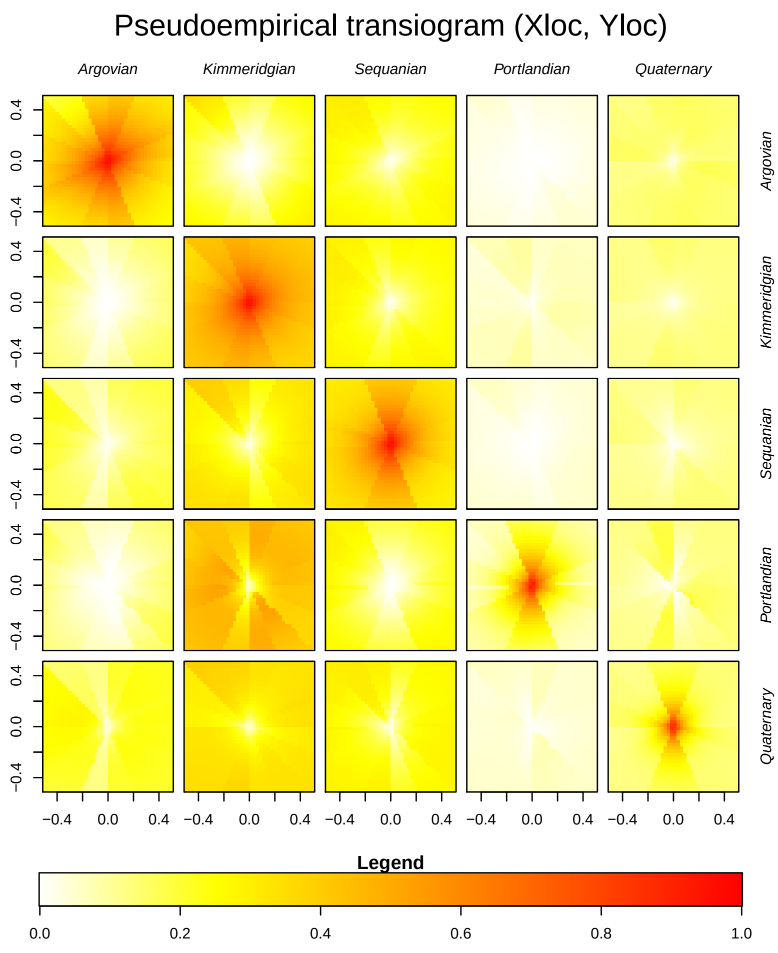


Figure 2: Multidimensional pseudoempirical transiogram.

along one fixed direction ϕ, we need to look at empirical transition probabilities computed for large distances. If most of these probabilities reproduce the characteristics already described, the data might be considered as a realisation from a weakly stationary process.

The comparison can also be made between two or more transiograms drawn for different directions. In so doing, it is possible to check if the process is anisotropic. This happens when there is directional dependence in the data. From a probabilistic point of view, the transiogram may show different dynamics when it approaches the limit probability matrix along different directions. In our case, the behaviours of the empirical transition probabilities along the axial directions do not match. Although those based on the estimated model are more regular than the empirical, they are not similar. This means that the Jura data set is anisotropic.

The function `pemt()` can be considered as another tool to check the anisotropy of the process. It estimates the transition rate matrix for each multidimensional lag direction and computes the transition probabilities as in (1). At the same time the function calculates other probabilities through the transition rates computed as in (2). Then the probabilities are drawn by use of the function `image.pemt()` (see Figure 2). If probabilities at the same level (those with the same colour) are placed on ellipses, the process is characterised by geometrical anisotropy. Comparisons made by the use of `contour.pemt()` are more evident, because contour lines are displayed for both the pseudo-empirical and the theoretical probabilities in a unique graphic.

The following R code can be executed to obtain Figure 2:

```
psEmpTr <- pemt(data, coords, 40, max.dist = c(.5, .5))
mycol <- rev(heat.colors(500))
image(psEmpTr, col = mycol, useRaster = TRUE, breaks = c(0:500) / 500)
```

From a computational point of view, the model based on the ellipsoidal interpolation of transition rate matrices is the most efficient way to calculate transition probabilities given multidimensional lags. In these cases, the model coefficients can be separately estimated by a unique R function. Hence, the functions `multi_tpfit_ml()` and `multi_tpfit_me()` provide methods to estimate transition rate

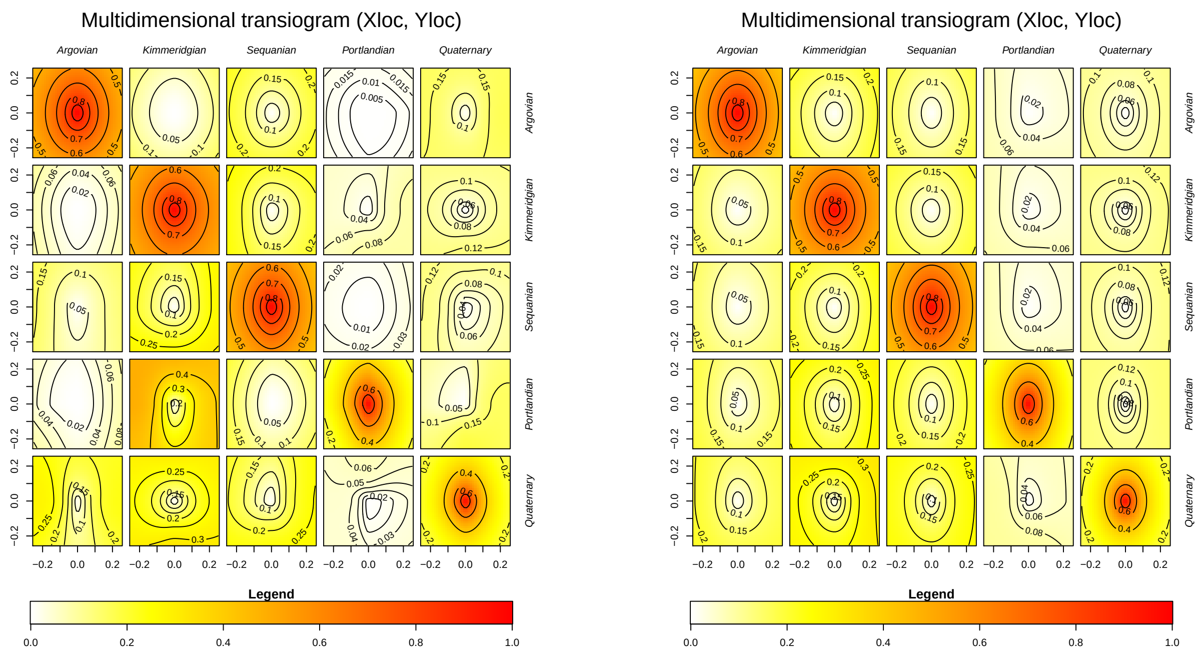


Figure 3: Multidimensional theoretical transiogram (transition rates are estimated by the average method (left) and by the maximum entropy method (right).

matrices along axial directions. These functions implement algorithms based on the mean lengths and the maximum entropy respectively (see Table 1).

```
MTr <- list()
MTr$average <- multi_tpfit_ml(data, coords)
MTr$entropy <- multi_tpfit_me(data, coords)
```

With the output of these functions, we can draw the theoretical transition probability maps as follows:

```
image(MTr$average, 40, max.dist = 0.25, col = mycol, nlevels = 5,
      breaks = 0:500 / 500)
image(MTr$entropy, 40, max.dist = 0.25, col = mycol, nlevels = 5,
      breaks = 0:500 / 500)
```

Both graphics in Figure 3 denote transition probability maps. The way to read these graphics is almost the same as for one-dimensional transiograms. Each image in this kind of graphic represents a 2-D transition probability section; this means that the probability level is given by the colour of the points. Each point is located to a specific "bidimensional" lag.

The transition probabilities obtained through maximum entropy rates (see Figure 3 on the right) are too regular for the process. If we look at the transiogram in Figure 1 on the left, we note that the red lines are not so close to the empirical transition probabilities and this may create some forecast problems when we consider the multidimensional lags. In fact, since the model was developed for stationary processes, its use is suitable when the stochastic process might be considered stationary.

Once the best fitting mdoel is chosen, we can predict the category in the unknown points or simulate a random field. In any case, we need to consider that the approximation of the simulation probabilities is affected by further variability due to the ellipsoidal interpolation of the transition rates. This means that the real transition rates for a non-axial direction can be overestimated or underestimated with a bigger error than the axial directions.

In this example, 100 observations are sampled from the original geological map and, instead of re-estimating transition rates, we are going to predict the category in the original locations through the already estimated rates. From a computational viewpoint, this allows us to compare the prediction accuracy of the procedures exposed in Table 1.

In real applications, all data may be used to estimate the parameters of the model and get better predictions. Simulations should be used only for drawing scenarios for non-observed locations. Obviously, the most probable category in a location is the best prediction.

The following lines of code are executed to plot the Jura geological map (see Figure 4):

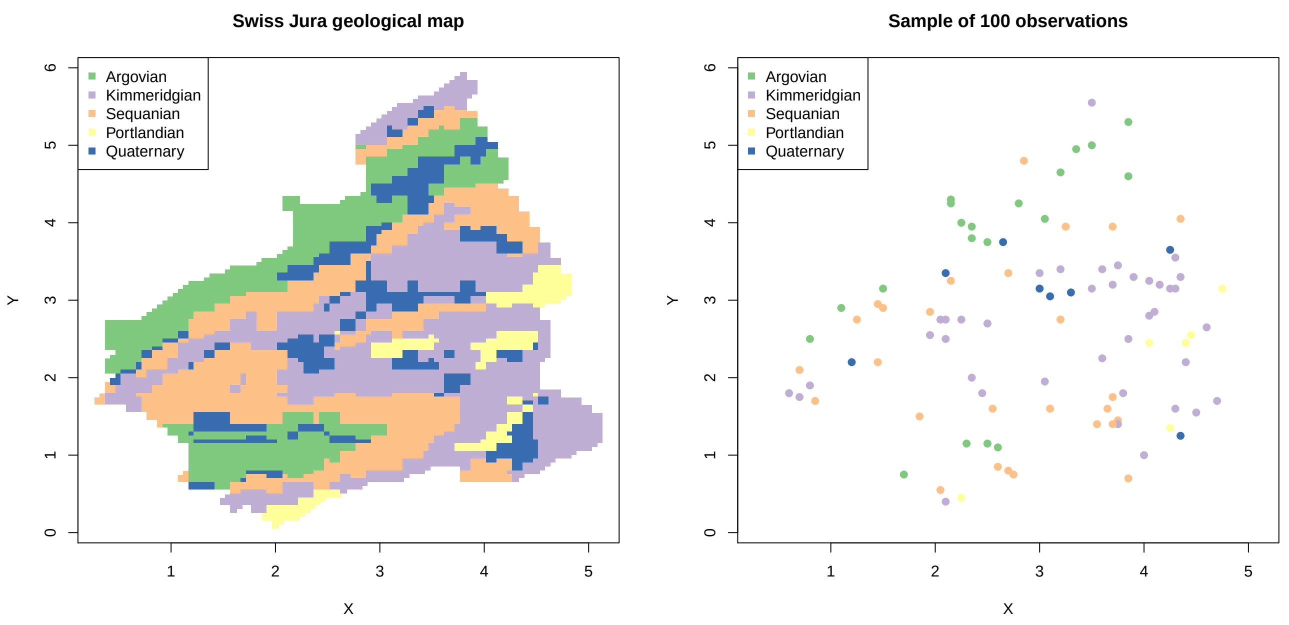


Figure 4: The Swiss Jura geological map (left) and the 100 sampled observations (right).

```
X <- jura.grid$X
Y <- jura.grid$Y
library(RColorBrewer)
brwCol <- brewer.pal(nlevels(data), "Accent")
par(mfrow = c(1, 1), mar = c(5, 4, 4, 2))
plot(X, Y, col = brwCol[data], pch = 15, main = "Swiss Jura geological map")
legend("topleft", legend = levels(data), col = brwCol, pch = 15)
```

One hundred observations are randomly selected as follows:

```
set.seed(29062011)
smp <- sample(length(data):1, 100)
```

and they are plotted by the following code:

```
plot(X, Y, type = "n", main = "Sample of 100 observations", xlab = "X", ylab = "Y")
points(X[smp], Y[smp], pch = 19, col = brwCol[data[smp]])
legend("topleft", legend = levels(data), col = brwCol, pch = 15)
```

Usually, before performing the simulation, a grid of points is generated by the use of the well-known expand.grid() function. In this example, the simulation grid is set as

```
grid <- jura.grid[, 1:2]
```

The kriging algorithm will approximate the conditional probabilities by considering the twelve nearest neighbours for all points in the simulation grid. Since only predictions will be presented here, the optimisation phase used to adjust the simulations will be skipped.

```
iks <- sim_ik(MTr$average,  data = data[smp], coords = coords[smp, ],
              grid, knn = 12, max.it = 0)
```

Both fixed and random path simulation methods are performed by considering those nearest points along the axial directions within a radius of length one.

```
fpth <- sim_path(MTr$average, data = data[smp], coords = coords[smp, ],
                 grid, radius = 1, TRUE)
rpth <- sim_path(MTr$average, data = data[smp], coords = coords[smp, ],
                 grid, radius = 1)
```

The multinomial categorical simulation method will approximate the prediction probabilities by considering all sample points.

```
mcs <- sim_mcs(MTr$average, data = data[smp], coords = coords[smp, ], grid)
```

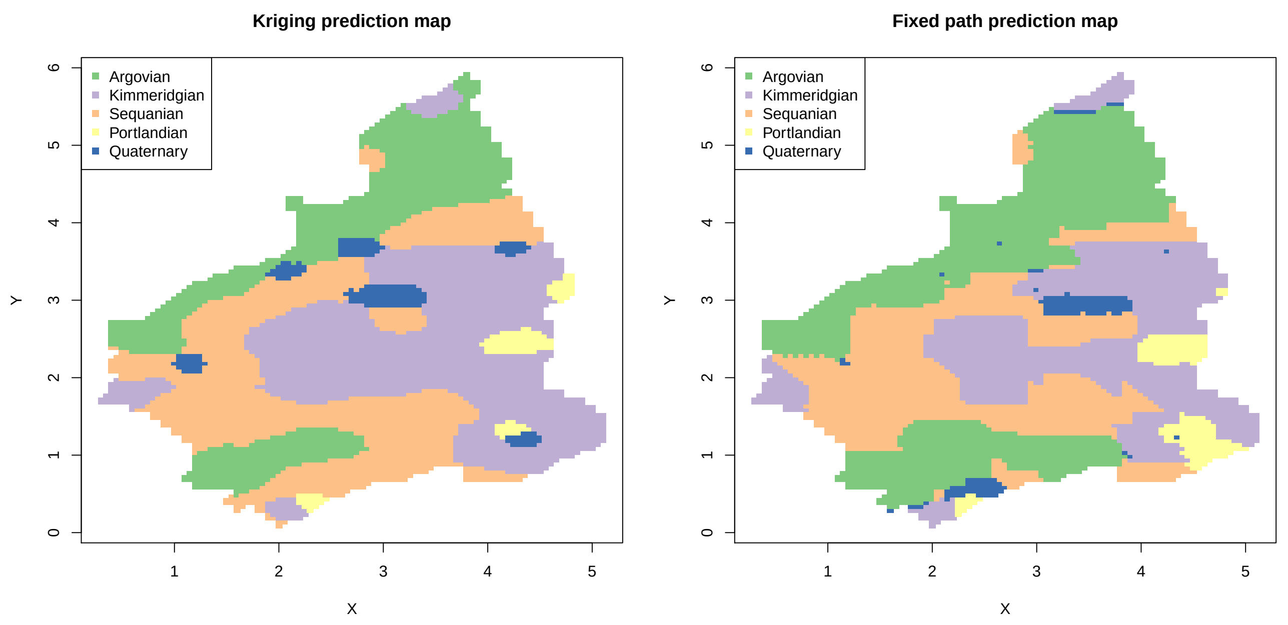


Figure 5: Prediction obtained by kriging probability approximation (left) and by fixed path probability approximation (right).

All these functions returns a "`data.frame`" object as output. It contains the coordinates, predictions, simulations and the approximated probability vector for each point in the simulation grid. Through these quantities we can plot the prediction maps by the use of the following R code:

```
posCol <- as.integer(iks$Prediction)
plot(X, Y, pch = 15, col = brwCol[posCol], main = "Kriging prediction map")
legend("topleft", legend = levels(data), col = brwCol, pch = 15)
posCol <- as.integer(fpth$Prediction)
plot(X, Y, pch = 15, col = brwCol[posCol], main = "Fixed path prediction map")
legend("topleft", legend = levels(data), col = brwCol, pch = 15)
posCol <- as.integer(rpth$Prediction)
plot(X, Y, pch = 15, col = brwCol[posCol], main = "Random path prediction map")
legend("topleft", legend = levels(data), col = brwCol, pch = 15)
posCol <- as.integer(mcs$Prediction)
plot(X, Y, pch = 15, col = brwCol[posCol],
     main = "Multinomial categorical prediction map")
legend("topleft", legend = levels(data), col = brwCol, pch = 15)
```

By looking at the graphics in Figures 5 and 6, we can have an idea of the prediction heterogeneity of these methods. In order to establish which is the best predictor, one should perform these simulations more than once. At each time, another 100 observations must be randomly selected. However, this is beyond the aim of this example.

Using only 2% of the original data, we can obtain the results in Table 2 by checking how many predictions match with the observed categories. Since the data are used in the computations of the probabilities, the prediction accuracy improves under particular conditions. Essentially, the sample size should increase while the spatial domain, wherein the observations are taken, is fixed and bounded. In this example, we can have more accurate predictions by increasing the number of the random selected observations and keeping the number of points in the simulation grid fixed.

In order to compute the number of matches, we calculate the contingency table as follows:

```
ikTb <- table(data, iks$Prediction)
fpTb <- table(data, fpth$Prediction)
rpTb <- table(data, rpth$Prediction)
mcTb <- table(data, mcs$Prediction)
```

The relative frequencies of matches are given by the following code:

```
ikPr <- sum(diag(ikTb)) / length(data)
fpPr <- sum(diag(fpTb)) / length(data)
```

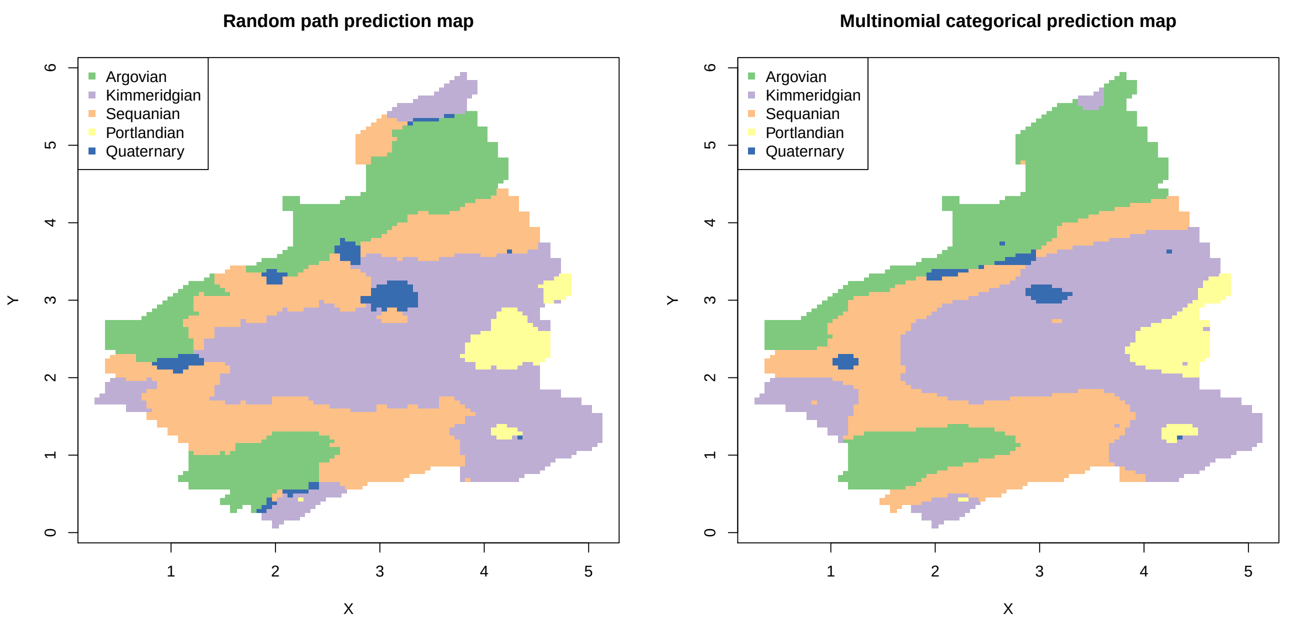

Random path prediction map
Multinomial categorical prediction map
Argovian
Kimmeridgian
Sequanian
Portlandian
Quaternary
Y
X
0
1
2
3
4
5
6

Figure 6: Prediction obtained by random path probability approximation (left) and by multinomial categorical probability approximation (right).

	Probability of coverage
Kriging	0.65452
Fixed path	0.54709
Random path	0.58922
Multinomial	0.60047

Table 2: Percentages of matched categories.

```
rpPr <- sum(diag(rpTb)) / length(data)
mcPr <- sum(diag(mcTb)) / length(data)
```

This allows us to obtain the values in Table 2.

Conclusions

Although there exist other approaches to study categorical random fields, the **spMC** package is a powerful tool for modelling continuous lag Markov chains. In particular, the user is allowed to deal with categorical response variables based on spatial stochastic processes without specifying a variogram model for the spatial dependence structure.

Several functions were developed to investigate graphically the properties of the process (e.g. stationarity and anisotropies), while others are useful to estimate the parameters, to predict and simulate the categorical values on unsampled locations. All of the functions in the package were designed in order to achieve good results in a reasonable time, as well as for large data sets.

Acknowledgements

I would like to thank the reviewers for providing useful comments and suggestions.

Bibliography

D. Allard, D. D'Or, and R. Froidevaux. An efficient maximum entropy approach for categorical variable prediction. *European Journal of Soil Science*, 62(3):381–393, 2011. doi: 10.1111/j.1365-2389.2011.01362.x. [p16, 19]

P. Bogaert. Spatial prediction of categorical variables: The Bayesian maximum entropy approach. *Stochastic Environmental Research and Risk Assessment*, 16:425–448, 2002. URL http://www.springerlink.com/content/c0d0cgfmrmflqpp4. [p20]

S. F. Carle. Implementation schemes for avoiding artifact discontinuities in simulated annealing. *Mathematical Geology*, 29(2):231–244, 1997. URL http://link.springer.com/article/10.1007%2FBF02769630. [p20]

S. F. Carle. *T-PROGS: Transition Probability Geostatistical Software*, 1999. URL http://gmsdocs.aquaveo.com/t-progs.pdf. [p16]

S. F. Carle and G. E. Fogg. Transition probability-based indicator geostatistics. *Mathematical Geology*, 28 (4):453–476, 1996. URL http://www.springerlink.com/content/xp60436q01g76m7j/. [p16, 19, 20]

S. F. Carle and G. E. Fogg. Modeling spatial variability with one and multidimensional continuous-lag Markov chains. *Mathematical Geology*, 29(7):891–918, 1997. URL http://www.springerlink.com/content/l8330v823535745j/. [p16]

O. F. Christensen and P. J. Ribeiro Jr. geoRglm: A package for generalised linear spatial models. *R News*, 2(2):26–28, 2002. URL http://CRAN.R-project.org/doc/Rnews. ISSN 1609-3631. [p16]

A. R. Conn, N. I. M. Gould, and P. L. Toint. A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds. *SIAM Journal of Numerical Analysis*, 28 (2):545–572, 1991. URL http://www.jstor.org/stable/2157828. [p18]

L. A. Goodman. The analysis of cross-classified data: Independence, quasi-independence, and interaction in contingency tables with and without missing entries. *Journal of the American Statistical Association*, 63:1091–1131, 1968. URL http://www.jstor.org/stable/2285873. [p18]

P. Goovaerts. *Geostatistics for Natural Resources Evaluation*. Applied Geostatistics Series. Oxford University Press, 1997. ISBN 9780195115383. [p16, 20]

N. J. Higham. *Functions of Matrices: Theory and Computation*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008. ISBN 9780898717778. [p16]

W. Li. A fixed-path Markov chain algorithm for conditional simulation of discrete spatial variables. *Mathematical Geology*, 39(2):159–176, 2007a. URL http://www.springerlink.com/content/f120u5h45u147t1k/. [p16]

W. Li. Markov chain random fields for estimation of categorical variables. *Mathematical Geology*, 39: 321–335, 2007b. URL http://www.springerlink.com/content/d0j5870n46k80351. [p19]

W. Li and C. Zhang. A random-path Markov chain algorithm for simulating categorical soil variables from random point samples. *Soil Science Society of America Journal*, 71(3):656–668, 2007. URL https://dl.sciencesocieties.org/publications/sssaj/pdfs/71/3/656. [p16]

J. R. Norris. *Markov Chains*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1998. ISBN 9780521633963. [p17]

OpenMP Architecture Review Board. OpenMP application program interface, version 3.0, May 2008. URL http://www.openmp.org/mp-documents/spec30.pdf. [p16]

E. J. Pebesma. Multivariable geostatistics in S: The gstat package. *Computers & Geosciences*, 30(7): 683–691, 2004. URL http://www.sciencedirect.com/science/article/pii/S0098300404000676. [p16]

D. K. Pickard. Unilateral Markov fields. *Advances in Applied Probability*, 12(3):655–671, 1980. URL http://www.jstor.org/stable/1426425. [p19]

L. Sartore. *spMC: Continuous Lag Spatial Markov Chains*, 2013. URL http://CRAN.R-project.org/package=spMC. R package version 0.3.1. [p16]

M. Schlather. *RandomFields: Simulation and Analysis of Random Fields*, 2013. URL http://CRAN.R-project.org/package=RandomFields. R package version 2.0.66. [p16]

L. A. Smith. Mixed mode MPI/OpenMP programming. UK High-End Computing Technology Report, 2000. URL http://citeseerx.ist.psu.edu/viewdoc/download?rep=rep1&type=pdf&doi=10.1.1.97.398. [p16]

The MPI Forum. MPI: A message passing interface, 1993. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.52.5877&rep=rep1&type=pdf. [p16]

Luca Sartore
Dipartmento di Statistica
Università degli Studi di Padova
Padova, Italia
sartore@stat.unipd.it

RNetCDF – A Package for Reading and Writing NetCDF Datasets

by Pavel Michna and Milton Woods

Abstract This paper describes the **RNetCDF** package (version 1.6), an interface for reading and writing files in Unidata NetCDF format, and gives an introduction to the NetCDF file format. NetCDF is a machine independent binary file format which allows storage of different types of array based data, along with short metadata descriptions. The package presented here allows access to the most important functions of the NetCDF C-interface for reading, writing, and modifying NetCDF datasets. In this paper, we present a short overview on the NetCDF file format and show usage examples of the package.

Introduction

NetCDF is a widely used file format in atmospheric and oceanic research – especially for weather and climate model output – which allows storage of different types of array based data, along with a short data description. The NetCDF format (Network Common Data Format, `http://www.unidata.ucar.edu/software/netcdf/`) has been developed since 1988 by Unidata (a programme sponsored by the United States National Science Foundation) with the main goal of making best use of atmospheric and related data for education and research (Rew et al., 2011; Rew and Davis, 1990).

NetCDF files are stored as machine-independent binary data, such that files can be exchanged between computers without explicit conversion (Rew and Davis, 1990). Until version 3.6.0, only one binary data format was used. This is the default format for all NetCDF versions and is also named NetCDF classic format (Rew et al., 2011). Version 3.6.0 of the NetCDF library introduced the 64-bit offset format, which allowed addressing of much larger datasets; version 4.0.0 introduced also the HDF5 format, in a way that the NetCDF library can use HDF5 as its external format. By default, however, the classic format is still used (Rew et al., 2011).

One particular advantage of NetCDF over some other binary formats, such as the RData format used by R, is the ability to access and modify arbitrary sections of array data. This allows massive datasets to be processed efficiently, even if they are larger than the virtual memory available on a particular system. To reduce disk space requirements, floating-point values are often packed into 8- or 16-bit integers, and the NetCDF-4 (HDF5) format supports transparent compression using the zlib library.

RNetCDF (Michna, 2012) was designed to handle the classic format and is also able to read and write files with 64-bit offset. If **RNetCDF** is compiled and linked with version 4.0.0 or later of the NetCDF library, files in NetCDF-4 (HDF5) binary format can be read if they use the data model of NetCDF-3 or earlier. In this paper we give a short overview of the concept of NetCDF based on Unidata's reference manuals, followed by the concept of the package and usage examples.

What are NetCDF datasets?

Data model

A NetCDF dataset contains dimensions, variables, and attributes, each identified both by a name and an ID number. These components can be used together to capture the meaning of data and relations among data fields in an array-oriented dataset. The NetCDF library allows simultaneous access to multiple NetCDF datasets which are identified by dataset ID numbers, in addition to ordinary file names.

A NetCDF dataset contains a symbol table for variables containing their name, data type, rank (number of dimensions), dimensions, and starting disk address. Each element is stored at a disk address which is a linear function of the array indices (subscripts) by which it is identified. Hence, these indices need not be stored separately (as in a relational database). This provides a fast and compact storage method (Rew et al., 2006). The advantage of the NetCDF library is that there is no need for the user to take care of the physical representation of multidimensional data on the disk.

Name	Length	Description	Limits		
`char`	8-bit	characters intended for representing text			
`byte`	8-bit	signed or unsigned integers	-128	...	$+127$
`short`	16-bit	signed integers	$-32'768$	...	$+32'767$
`int`	32-bit	signed integers	$-2'147'483'648$	...	$+2'147'483'647$
`float`	32-bit	IEEE floating-point (6 significant digits)	$\pm 1.175 \times 10^{-38}$	...	$\pm 3.403 \times 10^{38}$
`double`	64-bit	IEEE floating-point (15 significant digits)	$\pm 2.225 \times 10^{-308}$	...	$\pm 1.798 \times 10^{308}$

Table 1: External data types which are supported by the NetCDF interface.

Data types

The NetCDF interface defines six primitive external data types – `char`, `byte`, `short`, `integer`, `float`, and `double` (Rew et al., 2006). Their exact representation is shown in Table 1. These types were chosen to provide a reasonably wide range of trade-offs between data precision and number of bits required for each value. These external data types are independent of whatever internal data types are supported by a particular machine and language combination. The basic unit of named data in a NetCDF dataset is a variable (Rew et al., 2006). When a variable is defined, its shape is specified as a list of dimensions. These dimensions must already exist at the time of definition of a variable.

Dimensions

A dimension may be used to represent a real physical dimension, for example, time, latitude, longitude, or height. A dimension might also be used to index other quantities, for example station or model-run-number (Rew et al., 2006).

A NetCDF dimension has both a name and a length, where the dimension length is an arbitrary positive integer starting at 1. One dimension in a NetCDF dataset can be of unlimited length. Such a dimension is called the unlimited dimension or the record dimension. A variable with an unlimited dimension can grow to any length along that dimension. The unlimited dimension index is like a record number in conventional record-oriented files. A NetCDF dataset can have at most one unlimited dimension, but need not have any. If a variable has an unlimited dimension, that dimension must be the most significant (slowest changing) one.

Variables

Variables are used to store the bulk of the data in a NetCDF dataset. A variable represents an array of values of the same type. A scalar value is treated as a 0-dimensional array. A variable has a name, a data type, and a shape described by its list of dimensions specified when the variable is created. A variable may also have associated attributes, which may be added, deleted or changed after the variable is created (Rew et al., 2006). The shape of a variable cannot be changed after definition, only growing along the unlimited dimension is possible. Missing values (NA) have no internal representation. For this purpose, a respective attribute has to be defined for each variable. Most applications (including **RNetCDF**) accept the names '`_FillValue`' and '`missing_value`', although the latter is deprecated and should not be used when creating new datasets.

Handling of strings

NetCDF does not have a primitive string type, but does have arrays of type `char`, each of which is 8 bits in size. The main difference is that strings are arrays of chars of variable length, while char arrays are of fixed length (Rew et al., 2006). If an array of strings has to be created (e.g., a list of station names), internal routines read char arrays and convert them to strings without requiring the user to deal with trailing zeroes or padding. The zero-byte termination of strings is done automatically, and the user only needs to ensure that the fastest varying dimension (often named '`max_string_length`') is long enough to contain the terminating zero-byte, i.e., `max(nchar(my_strings))+1` where `my_strings` is the data to be written.

Attributes

NetCDF attributes are used to store metadata, similar in many ways to the information stored in data dictionaries and schema in conventional database systems. Most attributes provide information about a specific variable. These are identified by the name (or ID) of that variable, together with the name of the attribute. An attribute has an associated variable (the null variable for global attributes), a name, a data type, a length, and a value (Rew et al., 2006). Most generic applications that process NetCDF datasets assume standard attribute conventions (see below) and it is strongly recommended that these be followed unless there are good reasons for not doing so.

Naming conventions

There are almost no restrictions on how a NetCDF dataset should be named and structured. However, there are different conventions like COARDS and CF (`http://cf-pcmdi.llnl.gov/`, Eaton et al. 2011), and it is highly recommended to follow at least the basic practice to ensure portability and self-description of the contents. Variable, dimension and attribute names should begin with a letter and be composed of letters (case significant), digits, and underscores. The CF-convention (NetCDF Climate and Forecast Metadata Convention) permits neither the use of the hyphen character, nor leading underscores in names. Finally, NetCDF files should have the file name extension '`.nc`'.

Coordinate systems

A *coordinate variable* is a one-dimensional variable with the same name as a dimension, which names the coordinate values of the dimension. It should not contain any missing data (for example, no '`_FillValue`' or '`missing_value`' attributes) and must be strictly monotonic (values increasing or decreasing). A variable's *coordinate system* is the set of coordinate variables used by the variable. It is good practice to respect the following rules (Rew et al., 2006):

- Create coordinate variables for every dimension (except for string length dimensions).
- Give each coordinate variable at least '`unit`' and '`long_name`' attributes to document its meaning.
- Share dimensions to indicate that two variables use the same coordinates along that dimension. If two variables' dimensions are not related, create separate dimensions for them, even if they happen to have the same length.

In climatological applications, often geographical coordinates are used. Variables representing latitude must always explicitly include the '`units`' attribute; there is no default value. The recommended unit of latitude is '`degrees_north`', and '`degrees_east`' for longitude (Eaton et al., 2011).

Handling of time

There is no single way to deal with time in NetCDF datasets, but in most cases, time definitions from Unidata's UDUNITS library are used (see `http://www.unidata.ucar.edu/software/udunits`). Variables representing time must always explicitly include the '`units`' attribute; there is no default value. The '`units`' attribute takes a string value formatted as per the recommendations in the UDUNITS package (Eaton et al., 2011), usually in the form '`time_units since time_reference`'.

The most commonly used time units (and their abbreviations) include '`day`', '`hour`', '`minute`' and '`second`' or their plural forms. The units '`year`' and '`month`' may also be used, but they refer to fractional numbers of days related to successive passages of the sun through the vernal equinox. It may be preferable to use units related to the calendar year, including a '`common_year`' of 365 days, a '`leap_year`' of 366 days, a '`Julian_year`' of 365.25 days, or a '`Gregorian_year`' of 365.2425 days.

A reference time string is required to appear after the identifier '`since`', and it may include date alone, date and time, or date, time and time zone. An example of a valid reference time is '`1970-1-1 00:00:00 10:00`', which is midnight on January 1st, 1970 in a time zone that is 10 hours east of Coordinated Universal Time (such as Australian Eastern Standard Time).

Implementation

RNetCDF enables most of the functionality of the NetCDF C-interface (version 3.6.1) to be called from within R. Because time is often stored as a numeric vector with a reference time and unit according to Unidata's UDUNITS library, calendar conversion functions from UDUNITS are also included in the package.

The programming interfaces provided by **RNetCDF** will be familiar to developers who have used NetCDF from compiled languages such as Fortran and C. An alternative package, **ncdf** (Pierce, 2011) and its successor **ncdf4** (Pierce, 2013), provides a higher-level interface to NetCDF that may be preferred by some users, but it does not allow deleting and renaming of attributes. However, the lower-level functions in **RNetCDF** allow users to define functions and data structures that match their purposes. Although a high-level interface generally requires less work by users, we believe that **RNetCDF** provides more and better functionality, since users need not care about technical issues at the C level, yet they still have the means to perform nearly all operations that are possible on NetCDF datasets. We have included one higher-level function that is not part of the C-interface which enables reading of a whole NetCDF dataset using one command, which is a common task when working with such datasets.

All six of the external data types shown in Table 1 are supported. However, when reading data into R, only the R data types `character` and `numeric` will be distinguished. The NetCDF C-library converts integer and floating point values in a NetCDF file to double precision values in R, and the reverse conversions are performed during write operations. Reading and writing of data arrays is done by specifying a corner and a vector of edge lengths. The capabilities of the package are restricted to consecutive read/write; subsampling and mapping are not currently supported by **RNetCDF** but they can be performed easily using standard R commands.

The classic and 64-bit NetCDF file formats store metadata, such as dimensions, variable names and attributes, in a binary header at the start of the file. The contents of variables are stored after the header, and little or no padding is used to separate the sections of the file. If the metadata of the file are changed after variables are written, it is likely that variables will need to be moved within the file to accommodate a change in the size of the header. To avoid the overhead of such data movement, the usual approach is to define all of the metadata before writing data to variables. The NetCDF C-library uses distinct modes for defining metadata and writing variables, and special routines are used to switch between these two modes. However, for the sake of simplicity, the mode-switching routines are hidden by the **RNetCDF** interface, and the appropriate mode is selected for each operation requested.

Usage examples

Creating a NetCDF dataset

As an example, assume we have a climatological dataset with daily temperature measurements at five imaginary weather stations. Three variables are defined: time (as date with year, month, day, hour, minute, second), temperature and station name:

```
mytime        <- matrix(nrow=2, ncol=6)
mytime[1,]    <- c(2012, 06, 01, 12, 00, 00)
mytime[2,]    <- c(2012, 06, 02, 12, 00, 00)
mytime_units  <- "days since 1970-01-01 00:00:00"

mytemperature <- matrix(c(1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, NA, NA, 9.9),
                        ncol=2, nrow=5)
myname        <- c("Alfa", "Bravo", "Charlie", "Delta", "Echo")
```

When creating the NetCDF dataset, the organisation of the data should be known in advance. While changes to the structure of the file are possible, they may involve significant reorganisation of data within the file. To allow for expansion of a file with new data, it is possible to declare a single dimension with "unlimited" size. As a first step in our example, the file has to be created and all dimensions and variables need to be defined:

```
nc <- create.nc("foo.nc")

dim.def.nc(nc, "station", 5)
dim.def.nc(nc, "time", unlim=TRUE)
dim.def.nc(nc, "max_string_length", 32)

var.def.nc(nc, "time", "NC_INT", "time")
var.def.nc(nc, "temperature", "NC_DOUBLE", c("station", "time"))
var.def.nc(nc, "name", "NC_CHAR", c("max_string_length", "station"))
```

At this point, missing values (NA) cannot be written and the time axis is not yet defined. For this purpose, attributes have to be set and the time matrix needs to be converted into a vector with a reference time (as defined already above):

```
att.put.nc(nc, "temperature", "_FillValue", "NC_DOUBLE", -99999.9)
att.put.nc(nc, "time", "units", "NC_CHAR", mytime_units)
```

```
mytime_ut <- utinvcal.nc(mytime_units, mytime)
```

Now the variable data can be written. To ensure that the data are written to the file and not buffered in memory, the file should be closed when all operations are complete:

```
var.put.nc(nc, "name", myname)
var.put.nc(nc, "time", mytime_ut)
var.put.nc(nc, "temperature", mytemperature)
close.nc(nc)
```

If more data is to be added to the file in the same R session, the file may be left open, but to avoid loss of data, it may be desirable to force the flushing of buffers to disk using the function `sync.nc()` at critical stages of a calculation.

In our example, the NetCDF dataset is written to disk with the absolute minimum of required attributes. However, such a dataset is not really self-describing and would not conform with any conventions. Therefore, further attributes would need to be set. According to the CF-standard, a variable should have at least the attributes '`long_name`' (e.g., '`measured air temperature`'), '`units`' (e.g., '`degrees_celsius`'), and '`standard_name`' (e.g., '`air_temperature`') (the latter is not needed for the time coordinate variable). The possible values for '`standard_name`' can be found in the CF conventions document. CF also requests the indication of six global attributes, namely '`title`', '`history`', '`institution`', '`source`', '`comment`', and '`references`'. Although not mandatory, it is recommended that NetCDF datasets comply with the CF or any other standard, so that the contents of a file are described unambiguously. If these rules are followed, NetCDF datasets can be explored and processed using general-purpose software, and they can be distributed or archived without any risk that the data in a file could become separated from its description.

Reading an existing NetCDF dataset

To show the contents of a NetCDF dataset, it must first be opened with the `open.nc()` function. The `print.nc()` function displays an overview of the dataset on standard output, giving the dimension definitions, variable definitions including their attributes, and the contents of the global attributes. For the example dataset created earlier, an overview can be displayed as follows:

```
nc <- open.nc("foo.nc")
print.nc(nc)
```

The contents of a single variable can be read from a NetCDF dataset using the `var.get.nc()` function. For a variable that contains a large array of data, it may be desirable to read only an array section from the variable, which can be accomplished by specifying a start index and number of elements for each dimension of the array, as demonstrated below. Notice that the optional `start` and `count` arguments are vectors with one element for each dimension. Where the `count` argument has a value of NA, the corresponding dimension is read in full.

```
mytemp <- var.get.nc(nc, "temperature", start=c(NA,2), count=c(NA,1))
```

The easiest way to read the contents of all variables from a NetCDF dataset is by using the function `read.nc()`, which is available in **RNetCDF** version 1.6 or later. This function returns a list with the variables as named elements. Although this function has no equivalent in the NetCDF C-interface, it has been added to **RNetCDF** to simplify a common operation. For example, the contents of all variables can be read from our example dataset and the '`temperature`' variable copied to another variable using the following commands:

```
nc_data <- read.nc(nc)
mytemp  <- nc_data$temperature
```

Attributes can be read from variables that are identified by name or number, and global attributes can be read using the special variable name '`NC_GLOBAL`'. For example, conversion of relative times into calendar times requires the '`units`' attribute from the '`time`' variable, which may be read using the `att.get.nc()` function:

```
time_units <- att.get.nc(nc, "time", "units")
```

The NetCDF C-library provides a comprehensive set of functions to determine the structure of a NetCDF dataset, including the names and sizes of dimensions and variables. These functions can be used to write programs that handle NetCDF datasets without prior knowledge of their contents. Most of the inquiry functions of the C-library are accessible through the **RNetCDF** functions `file.inq.nc()`, `dim.inq.nc()`, `var.inq.nc()` and `att.inq.nc()`, which provide detailed information about datasets, dimensions, variables and attributes, respectively. For example, the names of all dimensions in a NetCDF dataset can be determined as shown below. Note that NetCDF dimensions can be referenced by integers that are sequential from 0; the same applies to variables and attributes.

```
ndims    <- file.inq.nc(nc)$ndims
dimnames <- character(ndims)
for(i in seq_len(ndims)) {
    dimnames[i] <- dim.inq.nc(nc, i-1)$name
}
```

Packed variables

To reduce the space required for storage of NetCDF datasets, the CF-convention allows variables to be stored in a packed format. The values are stored in a variable with lower precision than the original data. For example, 32-bit floating point values are often converted to 16-bit integers, so that the file size is approximately halved. To minimise the loss of information caused by the conversion, the original values are shifted and scaled so that they span the range of the new data type.

The packing algorithm can be expressed as follows:

$$\begin{aligned} x_s &= (\max x - \min x)/(\max y - \min y) && (1)\\ x_o &= \min x - x_s \min y && (2)\\ y &= (x - x_o)/x_s, && (3) \end{aligned}$$

where x is the original data and y is the packed variable. The values of x_o and x_s are stored with the packed variable in the standard attributes '`add_offset`' and '`scale_factor`', respectively. These attributes allow the packing operation to be reversed, although the unpacked data will usually have less precision than the original values.

Versions 1.6 or later of **RNetCDF** provide options to convert packed variables during reading and writing. Functions `var.get.nc()` and `var.put.nc()` have optional arguments `unpack` and `pack` respectively, although they have default values of `FALSE` to ensure compatibility with previous versions. The newly released function **read.nc** also has an optional `unpack` argument, which has the default value of `TRUE` to provide easy access to most datasets. It should be noted that the `pack` and `unpack` options are only honoured for variables that define both of the attributes '`add_offset`' and '`scale_factor`'.

In the example considered previously, the temperature data could be stored in a packed variable during creation of the dataset as follows:

```
var.def.nc(nc,"temp_p","NC_SHORT", c("station", "time"))
att.put.nc(nc, "temp_p", "_FillValue", "NC_SHORT", -32767)

tmax   <- max(mytemperature, na.rm=TRUE)
tmin   <- min(mytemperature, na.rm=TRUE)
ymax   <-  32766
ymin   <- -32766
scale  <- (tmax-tmin)/(ymax-ymin)
offset <- tmin-ymin*scale

att.put.nc(nc, "temp_p", "add_offset", "NC_DOUBLE", offset)
att.put.nc(nc, "temp_p", "scale_factor", "NC_DOUBLE", scale)

var.put.nc(nc, "temp_p", mytemperature, pack=TRUE)
```

Calendar functions

The two calendar functions `utcal.nc()` and `utinvcal.nc()` of the package (converting time from arbitrary units into a UTC-referenced date and time, and vice versa) have the option to read/write date and time directly in string form. When reading such strings, the structure must be exactly '`YYYY-MM-DD hh:mm:ss`'.

```
> utcal.nc("days since 2012-01-01 00:00:00", c(0,1))

     year month day hour minute second
```

```
[1,] 2012     1   1    0      0      0
[2,] 2012     1   2    0      0      0
```

It is also possible to specify another timezone as the reference time, as shown in the following example using Central European Time (CET):

```
> utcal.nc("days since 2012-01-01 00:00 +01:00", c(0,1))

     year month day hour minute second
[1,] 2011    12  31   23      0      0
[2,] 2012     1   1   23      0      0
```

If a user needs to have the date and time information as a string, the `type` argument can be set appropriately:

```
> utcal.nc("days since 2012-01-01 00:00 +01:00", c(0,1), type="s")

[1] "2011-12-31 23:00:00" "2012-01-01 00:00:00"
```

This functionality is intended especially for extracting axis descriptions in an efficient manner. Formatting of the string is possible using R functions for strings. For example, `substr()` can be used to extract the date or time components of the time-stamp.

Summary and outlook

RNetCDF is an R interface to the NetCDF C-library. Most of the functions provided by version 3 of NetCDF are accessible through **RNetCDF** in a way that allows users to build functions easily for their specific needs. Some higher-level features for frequently used operations are provided by **RNetCDF**, such as automatic support for missing values and packed variables and the ability to read all variables into an R list. Calendar conversion functions from Unidata's UDUNITS library are also included in this package to simplify the handling of time variables in NetCDF datasets.

Further information can be obtained in the **RNetCDF** reference manual and help pages (available from CRAN), Unidata's documentation for NetCDF (`http://www.unidata.ucar.edu/software/netcdf/docs/`) and UDUNITS (`http://www.unidata.ucar.edu/software/udunits/`), and the CF conventions documentation site (`http://cf-pcmdi.llnl.gov/documents/`).

The plans for future development include an option to read and write POSIXt time variables, which are used by many R routines, with automatic translation to and from the time format used in NetCDF datasets. The next major update will include support for the extended data model of NetCDF-4. However, a first analysis of the full NetCDF-4/HDF5 data model revealed that it might be difficult to map user defined data types (e.g., 6-bit structures) in a straight-forward way in R, so an intensive analysis of the new data model and the requirements of R users will be needed.

Readers who are interested in contributing to the development of **RNetCDF** are invited to contact the authors.

Acknowledgements

Brian Ripley and Uwe Ligges are gratefully acknowledged for making the Windows version possible, Simon Urbanek for enabling a binary distribution for Mac OS X.

Bibliography

B. Eaton, J. Gregory, B. Drach, K. Taylor, and S. Hankin. *NetCDF Climate and Forecast (CF) Metadata Conventions, Version 1.6*, 2011. [p31]

P. Michna. *RNetCDF: R Interface to NetCDF Datasets*, 2012. URL `http://CRAN.R-project.org/package=RNetCDF`. R Package Version 1.6.1-2. [p29]

D. Pierce. *ncdf: Interface to Unidata netCDF data files*, 2011. URL `http://CRAN.R-project.org/package=ncdf`. R Package Version 1.6.6. [p31]

D. Pierce. *ncdf4: Interface to Unidata netCDF (version 4 or earlier) format data files*, 2013. URL `http://CRAN.R-project.org/package=ncdf4`. R Package Version 1.10. [p31]

R. Rew and G. Davis. NetCDF: An interface for scientific data access. *IEEE Computer Graphics and Applications*, 10(4):76–82, 1990. [p29]

R. Rew, G. Davis, S. Emmerson, H. Davies, and E. Hartnett. *The NetCDF Users Guide, Version 3.6.1*. Unidata Program Center, 2006. [p29, 30, 31]

R. Rew, G. Davis, S. Emmerson, H. Davies, E. Hartnett, and D. Heimbigner. *The NetCDF Users Guide, Version 4.1.3*. Unidata Program Center, 2011. [p29]

Pavel Michna
Institute of Geography
Hallerstrasse 12, CH-3012 Bern
Switzerland
michna@giub.unibe.ch

Milton Woods
Bureau of Meteorology
GPO Box 1289, Melbourne VIC 3001
Australia
M.Woods@bom.gov.au

Surface Melting Curve Analysis with R

by Stefan Rödiger, Alexander Böhm and Ingolf Schimke

Abstract Nucleic acid *Melting Curve Analysis* is a powerful method to investigate the interaction of double stranded nucleic acids. Many researchers rely on closed source software which is not ubiquitously available, and gives only little control over the computation and data presentation. R in contrast, is open source, highly adaptable and provides numerous utilities for data import, sophisticated statistical analysis and presentation in publication quality. This article covers methods, implemented in the **MBmca** package, for DNA Melting Curve Analysis on microbead surfaces. Particularly, the use of the second derivative melting peaks is suggested as an additional parameter to characterize the melting behavior of DNA duplexes. Examples of microbead surface Melting Curve Analysis on fragments of human genes are presented.

Introduction

Melting Curve Analysis

Nucleic acid *Melting Curve Analysis* (MCA) is a central step in nucleic acid[1] interaction studies, identification of specific DNA sequences after quantitative real-time PCR, genotyping or detection of *Single Nucleotide Polymorphisms*[2] (SNP) both in solution and on surfaces (Ririe et al., 1997; Gundry et al., 2003; Sekar et al., 2005; Zhou et al., 2005; Rödiger et al., 2012b). A review of the literature revealed that there is an ongoing demand for new bioanalytical devices to perform MCAs. This includes lab-on-chip systems or the recently published *VideoScan* platform (Rödiger et al., 2012b, see following section). Some of these systems offer software solutions for MCA but often custom made software is required. Although bioanalytical devices break new ground to meet criteria like high multiplex levels or new detection-probe-systems the fundamental concept of MCA remains unchanged.

The MCA is a real-time monitoring of a heat-induced double stranded nucleic acid dissociation which can be monitored by the change of the referenced mean/median fluorescence intensity (MFI)[3] at a defined temperature (Figure 1).

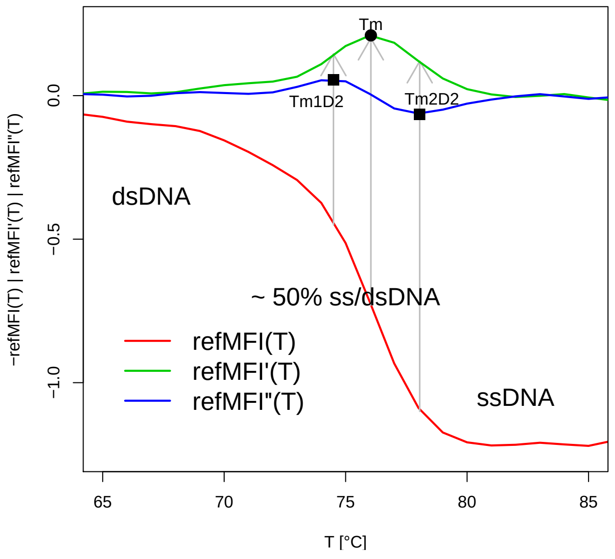


Figure 1: Melting curve and melting peaks of double stranded DNA. Raw data curves ($refMFI(T)$ —), approximate first derivative ($refMFI'(T)$ —) and second derivative ($refMFI''(T)$ —). Peak values: Tm is the maximum of the first derivative. Tm_1^{D2} and Tm_2^{D2} are the extremes (a minimum and a maximum) of the second derivative. dsDNA, double stranded DNA; ssDNA, single stranded DNA.

[1]Nucleic acids herein refer to deoxyribonucleic acids (DNA) and ribonucleic acids (RNA).

[2]*Single Nucleotide Polymorphisms* (SNP) are caused by exchanges of single nucleotides.

[3]In the past absorbency was the quantitative measure. New devices measure fluorescence intensities mediated by nucleic acid-intercalating fluorophores (e. g., *EvaGreen*®) or fluorophore labeled DNA probes (e. g., *Molecular Beacons*, Gašparič et al., 2010; Rödiger et al., 2012a). The values are often reported as RFU (Relative Fluorescence Units), MFI (Mean/Median Fluorescence Intensity) or $refMFI$ (Referenced Mean/Median Fluorescence Intensity).

By definition, the *melting point* (Tm) is the inflection point of the melting curve. On molecular level circa 50% of the nucleic acids are dissociated at Tm. The *melting peak* (Equation 1) can be determined from the first negative derivative (Equation 2) of the melting curve. At this temperature peak the rate of change is maximal. The Tm is highly reproducible, thus can be used as a "characteristic identity" to distinguish nucleic acid species.

$$Tm = \max(refMFI'(T)) \tag{1}$$

$$refMFI'(T) = -\frac{d(refMFI)}{d(T)} \tag{2}$$

To the best of our knowledge we are the first to suggest the peak values, designated Tm_1^{D2} and Tm_2^{D2} (Equation 3 and 4), of the second derivative (Equation 5) as additional measures for the characterization of nucleic acid melting processes on microbead surfaces. They show the maximal rate of change of $refMFI'(T)$. The corresponding temperature values (abscissa) at the inflection points of $refMFI'(T)$ occur where $refMFI''(T)$ reaches a (local) minimum and a (local) maximum, respectively (Figure 1). Both values offer additional quantitative measures to describe early and late phases of the melting process. The Tm_1^{D2} quantifies the maximal acceleration and Tm_2^{D2} the maximal deceleration of the dissociation process. The deceleration starts in the middle of the process, at Tm.

$$Tm_1^{D2} = \max(refMFI''(T)) \tag{3}$$

$$Tm_2^{D2} = \min(refMFI''(T)) \tag{4}$$

$$refMFI''(T) = -\frac{d^2(refMFI)}{d(T)^2} \tag{5}$$

Surface Melting Curve Analysis

In Rödiger et al. (2012b) we reported the *VideoScan* platform which provides a technology for various bioanalytical applications[4] with the focus on highly multiplex kinetic analysis. The *VideoScan* platform consists of a fully automated fluorescence microscope, a modular software package for data acquisition and a heating/cooling-unit (HCU)[5] for micro volume ($\leq 20\ \mu L$) samples. We developed temperature controlled assays to detect and analyze nucleic acid probes on the surface of microbeads.

In brief, different thermo-tolerant microbead populations, defined by varying ratios of two impregnated fluorophores, are included in the reaction. Each microbead population presents gene specific capture probes on their surfaces (Figure 2).

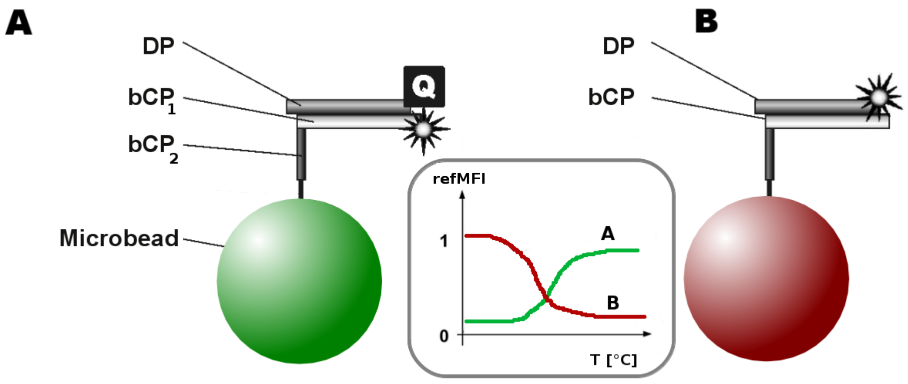


Figure 2: Principles of microbead *direct hybridization* probe systems. A) The *FRET assay* uses microbead bound capture probe (*bCP*) with the fluorophore- (*) and quencher-labeled (**Q**) detection probes (*DP*). A *bCP* consists of different regions (e. g., bCP_1, bCP_2) ready to hybridize with a complementary *DP*. B) The *standard assay* uses non-labeled *bCP*s which hybridize with fluorophore-labeled *DP*s. Inset) Both probe systems differ in the curve shape resulting from the melting process.

[4]These include autoimmune cell pattern recognition, microbead-based assay for DNA and proteins. For details see Willitzki et al. (2012) and Rödiger et al. (2012b).

[5]The heating/cooling-unit (HCU) is based on peltier elements and has a performance similar to conventional thermal cyclers (e. g., iQ5 (Bio-Rad Laboratories)). For details see Rödiger et al. (2012b).

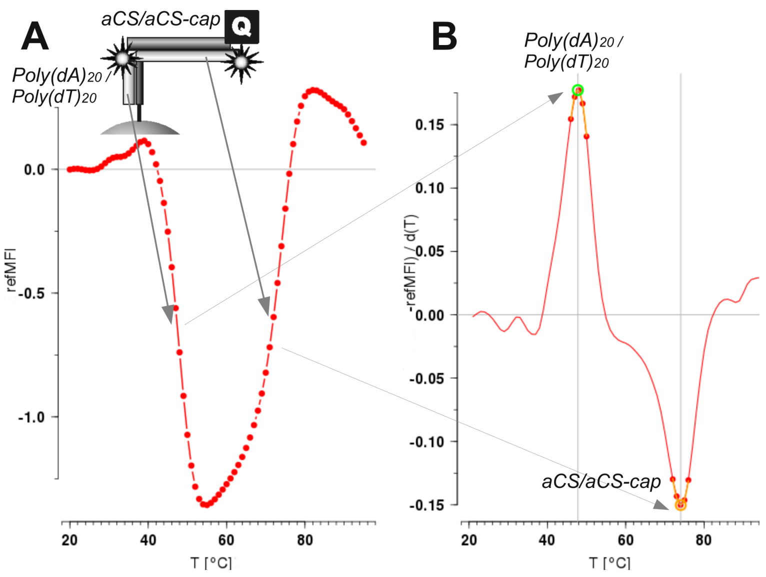


Figure 3: Melting curve and melting peaks. The data were obtained from a MCA experiment on microbead surfaces. Top left inset: Two DNA detection probes ($Poly(dA)_{20}$, aCS), were hybridized to a complementary microbead bound capture probe. A) While applying a temperature gradient (20 °C to 95 °C, 1 °C per step) the change of the $refMFI$ is monitored. B) This probe system exhibits two melting peaks. Analysis showed the first (positive) peak for $Poly(dA)_{20}$ (~48 °C) followed by a second (negative) peak of aCS (~74 °C).

Particularly, short 3′ or 5′ quencher/fluorophore-labeled probes were used[6]. The capture probes (bCP) are complementary to detection probe (DP) in solution (Figure 2 and inset Figure 13). Signals are mediated by temperature dependent dehybridization events between bCPs and DPs. All probe systems described here and our related works are characterized by one or two significant melting peaks at different temperatures and/or different signs (Figure 3). Due to different fluorophore / quencher combinations and probe systems (e. g., direct hybridization, dual-hybridization probes) the sign of a Tm peak value was designed to be either positive or negative (see Figure 3B). Transferred to applications in screening processes all subsequent analysis would be reduced to the identification of distinct Tms and their intensity of the corresponding microbead populations.

Why use R for Melting Curve Analysis?

In biomedical research many scientists rely on closed source software which gives only little control over the computation and data presentation. Most importantly reproduction of data and calculations from other research is limited or impossible. Just recently this was discussed in the two journals *Nature* and *Science* (Ince et al., 2012; Morin et al., 2012). Closed software tied to limited tasks, hinders the import of custom data and gives no control over or insight into the source code and therefore is not ideal for research. Basically any open computing language or tool can be used to overcome these issues. But R fulfills all requirements mentioned above. It is in an open system with numerous utilities for data import, processing, sophisticated statistical analysis and presentation. Many R packages are peer-reviewed and undergo an intensive testing. Most importantly R is open source and therefore methods or results can be reproduced independently.

One implementation for MCA with R is available from the excellent **qpcR** package by Ritz and Spiess (2008). The `meltcurve()` function provides a sophisticated method to process melting curve data from qPCR experiments in solution. It uses automatic optimization procedures to smooth and fit curves. This function is ideal for the identification of multiple Tms, the calculation of the peak area and high resolution data.

[6]The DNA probe sequences, microbeads, probe immobilization process and experimental set up were described in Rödiger et al. (2011, 2012b). The fluorescence dye *Atto 647N* and the quencher *BHQ2* were used generally. Intercalating dyes were not used since they are known to alter the melting process (Gudnason et al., 2007).

Motivation for the MBmca package

There is an ongoing interest to understand the melting behavior of nucleic acids on surfaces. Particularly the effects of the reaction environment, e. g., the microbead surface change due to functional groups or the density of *bCP* has not been investigated intensively for multiplex microbead assays. During the development of the *VideoScan* platform a simple and lightweight method for automatic screening, quality control and automatic detection of a limited number of melting peaks was needed. This article describes the **MBmca**[7] (Roediger, 2013) package which was developed as an approach for statistical MCA on microbeads. The ambition of the **MBmca** is not to compete with the **qpcR** or other R packages but rather to extend the scope of R for MCA on surfaces.

Implementation of MCA in the MBmca package

Functions and data sets of MBmca

The **MBmca** package includes the functions `MFIerror()`, `mcaPeaks()`[8] and `mcaSmoother()` for data inspection and preprocessing and `diffQ()`, `diffQ2()` for MCA. The data sets `DualHyb`, `DMP` and `MultiMelt` are raw fluorescence data measured with the *VideoScan* platform (Rödiger et al., 2012b) on microbead surfaces. The data are arranged as `data.frames` starting in the first column with the temperature (°C) followed by the fluorescence values (refMFI). The package has a dependency to **robustbase** (Rousseeuw et al., 2013) and uses mainly standard R functions. Elementary steps, e. g., data import, are similar to other R functions and thus used as described elsewhere (Venables et al., 2013).

Inspection of raw fluorescence data

One of the fundamental strengths of the MCA on microbead surfaces is the achievable multiplex level. The `MFIerror()` function was developed for a fast multiple comparison of the temperature dependent variance of $refMFI$. `MFIerror()` returns an object of the class `data.frame` with columns "Temperature", "Location" (Mean, Median), "Deviation" (Standard Deviation, Median Absolute Deviation) and "Coefficient of Variation".

- The argument `errplot` (default) sets `MFIerror()` to plot the results. In the default setting (`CV = FALSE`) the mean with the standard deviations is plotted. Using the argument `rob = TRUE` the median and the median absolute deviation (MAD) are plotted instead of the mean and standard deviation.
- If `CV` is true the coefficient of variation (CV) is plotted. Setting the argument `RSD = TRUE` shows the relative standard deviation (RSD) in percent.

In an example the mean raw fluorescence from multiplex melting curves of twelve microbead populations was evaluated for the probes *HPRT1* and $MLC-2v$ (`MultiMelt` data set). The probe system used corresponds to Figure 2A. Ideally the variance between the twelve microbead populations is low. `MFIerror()` takes the first column of `MultiMelt` as temperature value and columns 2 to 13 for the probes *HPRT1* and columns 14 to 25 for $MLC-2v$, respectively.

```
# Load MultiMelt data set.
data(MultiMelt)

# MFIerror for the HRPT1 data (column 2 to 13).
# The default settings of MFIerror show the the mean fluorescence and the
# standard deviation at a defined temperature.
MFIerror(MultiMelt[, 1], MultiMelt[, 2:13])

# MFIerror on the MLC-2v data (column 14 to 25).
MFIerror(MultiMelt[, 1], MultiMelt[, 14:25])
```

The corresponding plots are shown in Figure 4. The curves indicate that the different microbead populations show similar melting curve shapes but differ in height. At higher temperatures the values vary.

[7] **M**icro**B**ead **m**elting **c**urve **a**nalysis.

[8] `mcaPeaks()` is a function which can be used to estimate the number and location of the approximate local minima and maxima of melting curve data. This can be used to define a temperature range for MCA, melting curve quality control or peak height threshold definition (see Roediger 2013).

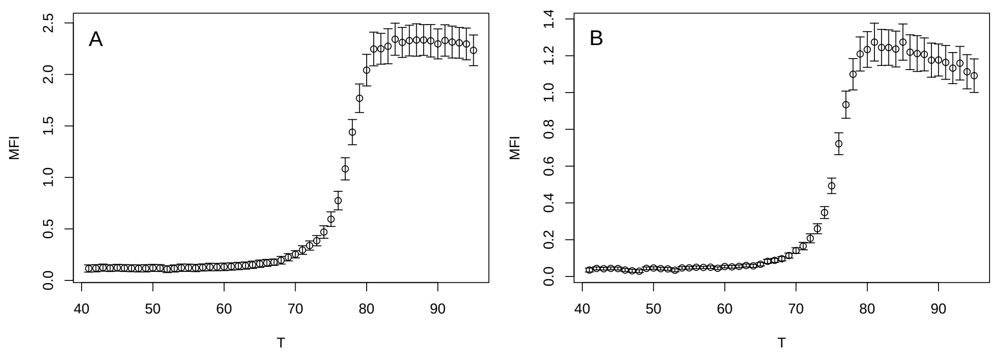


Figure 4: Inspection of the mean raw fluorescence from multiplex melting curves. `MFIerror()` with the argument `rob = FALSE` was used to compare the mean and the standard deviationof the temperature dependent fluorescence on twelve microbead populations for A) *HPRT1* and B) *MLC*−2*v*.

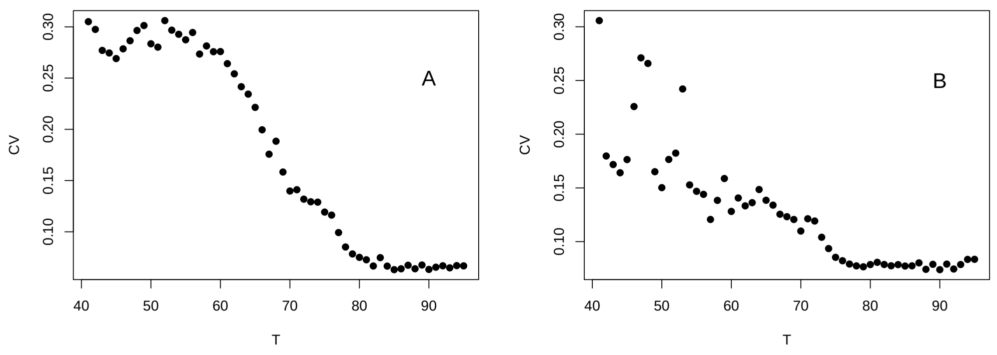


Figure 5: Comparison of the absolute coefficient of variation from multiplex melting curves. `MFIerror()` with argument `CV = TRUE` was used to plot the absolute coefficient of variation for twelve microbead populations with A) *HPRT1* or B) *MLC*−2*v*.

When `MFIerror()` is used with the argument `CV = TRUE` the coefficient of variation is presented (Figure 5). In the example all CV values are low (< 0.3) and even decrease with increasing temperatures for both *HPRT1* and *MLC*−2*v*.

We questioned how a single microbead population differs from the average of all microbead populations. The mean output of `MFIerror()`, with the argument `errplot = FALSE`, was subtracted by the fluorescence of *HPRT1* or *MLC*−2*v*. The results were assigned to `HPRT1.mean` and `MLC2v.mean`.

```
# Load MultiMelt data set.
data(MultiMelt)

# Use MFIerror to calculate the mean fluorescence for HRPT1 and MLC-2v over all
# twelve microbead populations.
HPRT1.mean <- MFIerror(MultiMelt[, 1], MultiMelt[, 2:13], errplot = FALSE)
MLC2v.mean <- MFIerror(MultiMelt[, 1], MultiMelt[, 14:25], errplot = FALSE)

# Draw figures on the graphics device in a 2x6 array
par(mfrow = c(2, 6))

# Calculate the difference between the fluorescence of a single microbead population
# and the average of all twelve microbead populations. Plot the results.
for (i in 1:12) {
  tmp.HPRT1 <- MultiMelt[, i + 1] - HPRT1.mean[, 2]
  tmp.MLC2v <- MultiMelt[, i + 13] - MLC2v.mean[, 2]
  plot(MultiMelt[, 1], tmp.HPRT1, main = paste("Pop", i, sep = ": "),
       pch = 19, ylim = c(-0.28, 0.28), xlab = "T", ylab = "delta")
  abline(h = 0, col = "black")
  abline(v = 65, col = "blue")
  points(MultiMelt[, 1], tmp.MLC2v, pch = 15, col = 2)
}
```

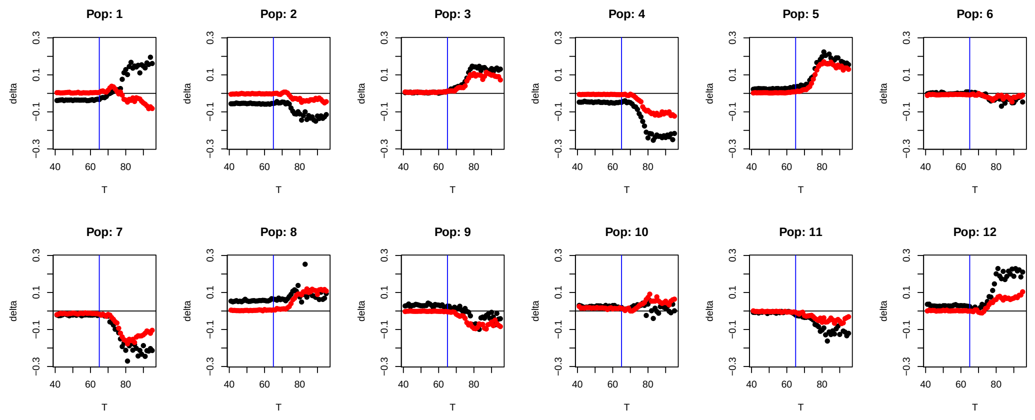


Figure 6: Difference plot of the fluorescence values. The temperature dependent mean fluorescence of twelve microbead populations, determined with `MFIerror()`, was subtracted from the fluorescence of single population ("Pop: "). • $HPRT1$, • $MLC{-}2v$, — base line, — start of the melt process.

Figure 6 shows low differences (delta) at temperatures below 65 °C due to near complete quenching. Above this temperature (start of the DNA probe strand dissociation) the differences to the mean fluorescence of all microbead populations grow. Apart from "Pop: 1" changes appear systematically which indicates that the bCP/DPs have a similar melting behavior on all microbead populations.

Preprocessing of raw fluorescence data

The differentiation is the central step of the MCA. Accessible textbook information confirms that differentiation may result in the amplification of noise. To reduce the noise, R provides numerous possibilities to fit smooth functions and use filter functions. Such operations may alter the curve shape considerably and thus lead to artificial results. Smooth functions available in R include the moving average (`filter()`, **stats**), the LOWESS smoother (`lowess()`, **stats**) which applies locally-weighted polynomial regression, fitting of a local polynomial regression (`loess()`, **stats**), Savitsky-Golay filter (`sgolayfilt()`, **signal**; The signal Developers, 2013), cubic splines (`smooth.spline()`, **stats**) or Friedman's SuperSmoother (`supsmu()`, **stats**). Although smoothed data might provide the impression of high quality data no guarantee for optimal results or that no peaks were artificially introduced is given. A good practice is to visualize the output combined with the original data. `mcaSmoother()` uses `smooth.spline()` and contains further helper function. `mcaSmoother()` should be used if the data may contain missing values, high noise or if the temperature resolution of the melting curve data is low ($\geq$ 0.5 °C / step) in order to correct the problems automatically.

- Measurements from experimental systems may occasionally include missing values (`NA`). Function `mcaSmoother()` uses `approx()`[9] to fill up `NA`s under the assumption that all measurements were equidistant. The original data remain unchanged and only the `NA`s are substituted.
- `mcaSmoother()` calls `smooth.spline()` to smooth the curve. Different strengths can be set using the argument `df.fact` (default 0.95). Internally it takes the degree of freedom value from the spline and multiplies it with a factor between 0.6 and 1.1. Values lower than 1 result in more strongly smoothed curves.
- If the argument `bgadj` is set `TRUE`, `bg` must be used to define a temperature range for a linear background correction[10]. The linear trend is estimated by a robust linear regression using `lmrob()`. In case criteria for a robust linear regression are violated `lm()` is used automatically.
- The argument `Trange` can be used to define a temperature range of the analysis.
- To scale the fluorescence a *Min-Max normalization* (Equation 6) between 0 and 1 can be used by setting the argument `minmax` to `TRUE`. This is useful if the fluorescence values between samples vary considerably, for example due to high background. An advantage of this normalization is the preservation of the relationships between the values. However, on surfaces normalization

[9]Besides `approx()` (**stats**) further functions are available from the **zoo** package (Zeileis and Grothendieck, 2005, e. g., `na.approx()`, `na.spline()`) and the **delftfews** package (Frasca, 2012, `na.fill()`, `na.interpolate()`). `approx()` was integrated since further dependencies are omitted and a linear interpolation is used.

[10]Some software packages include automatic linear or non-linear background correction which work not reliable in many cases. The examples in Figure 4 use identical probe systems (direct hybridization) but different bCP/DP combinations. In theory the melting curve shape should be very similar. Particularly values at temperatures above 80 °C vary strongly. Therefore only a simple linear background correction was implemented.

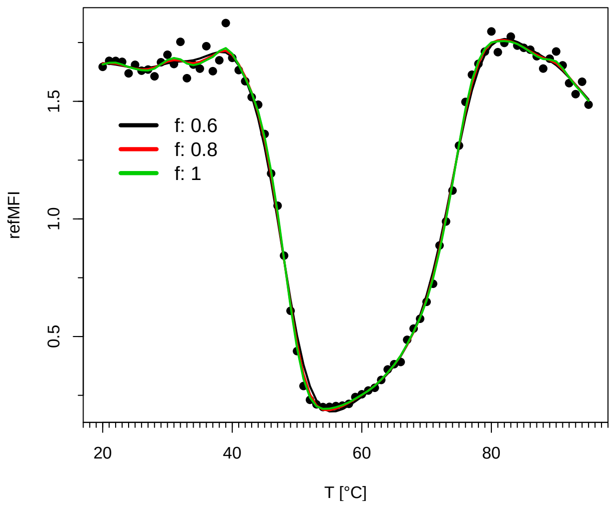


Figure 7: Raw fluorescence values (•) versus the temperature. Smoothed curves (f: 0.6 —, f: 0.8 —, f: 1 —) using `mcaSmoother()` with different strengths (f) to smooth the curves.

should be used with caution because it might lead to the false impression that all microbeads carried equal quantities of *bCP*.

- The argument `n` uses the `spline()` function to increase the temperature resolution of the melting curve data by n-times the length of the input temperature (see `mcaSmoother()` examples in Roediger 2013).

$$refMFInorm = \frac{refMFI - \min(refMFI)}{\max(refMFI) - \min(refMFI)} \quad (6)$$

`mcaSmoother()` returns an object of the class "`data.frame`" with the columns "x" (temperature) and "y" (fluorescence values). These can be used to plot the preprocessed data. For example, three arbitrary chosen strengths to smooth the curves (f: 0.6, 0.8, 1.0) were tested. Data from the `DMP` data set were used as follows:

```
# Load DMP data set.
data(DMP)

# Create plot with raw data.
plot(DMP[, 1], DMP[, 6], xlim = c(20, 95), xlab = "T [C]",
     ylab = "refMFI", pch = 19, col = 8)

# Add minor tick marks to the abscissa.
require(Hmisc); minor.tick(nx = 20)
```

The function `mcaSmoother()` is used in a loop to smooth the curve with user defined strengths (f). Wrapped into `lines()` it draws[11] the corresponding curves directly (Figure 7). In comparison to the original data a low filter strength (f: 1) is sufficient.

```
# Define three filter strengths (highest (0.6) to lowest (1.0)) and assign them
# to df.fact. Smooth the raw data and add the results as lines to the plot.
f <- c(0.6, 0.8, 1.0)
for (i in 1:3) {
  lines(mcaSmoother(DMP[, 1], DMP[, 6], df.fact = f[i]), col = i, lwd = 2)
}
# Add a legend to the plot with the filter strengths.
legend(20, 1.5, paste("f", f, sep = ": "), cex = 1.2, col =  1:3,
       bty = "n", lty = 1, lwd = 4)
```

In the next example `mcaSmoother()` was used with different arguments to (i) smooth the data, (ii) remove the background and (iii) to perform a Min-Max normalization. Data for *HPRT1* (Figure 8A) were taken from the `MultiMelt` data set. The plot of Figure 8B implies that it is sustainable to smooth

[11]A fine grained abscissa was created with `minor.tick()` from the **Hmisc** package (Harrell Jr et al., 2013) to enhance the plot.

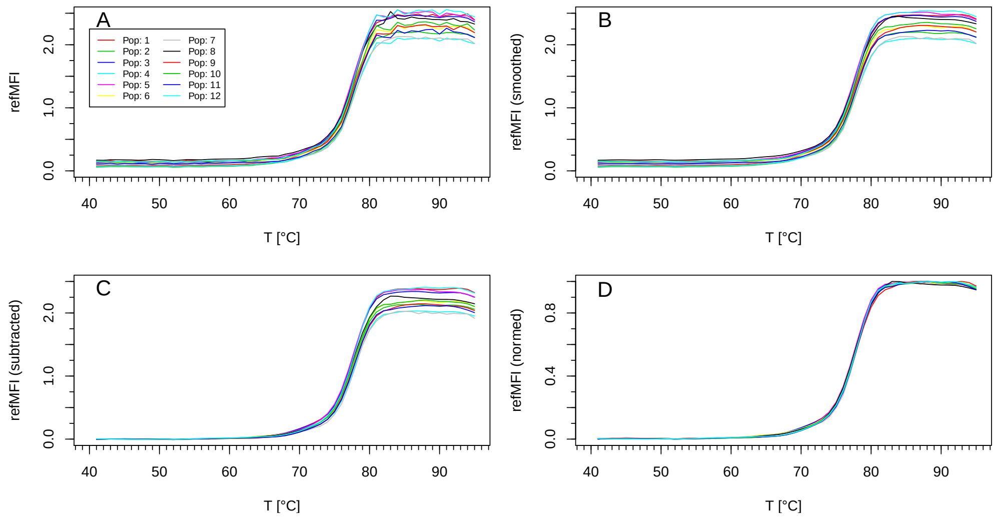


Figure 8: Preprocessing of *HPRT1* with `mcaSmoother()`. A) Raw fluorescence data of melting curves from 12 microbead populations ("Pop: "). B) Smoothed curves with the default settings (f: 0.95). C) The smoothed curves after background reduction (`bg = c(41, 61)`, i.e., 41 °C to 61 °C) and linear trend correction. D) Min-Max normalized curves.

the curve. Not immediately obvious, the twelve microbead populations have different final signal intensities. This is due to the different quantities of surface bound *bCP*s (not shown). However, this is easily visualized after a background correction (Figure 8C). All curves appear similar in shape after the Min-Max normalization. This indicates that there are no substantial differences between the microbead populations which obfuscates further MCAs (Figure 8D).

Calculation of the melting peaks

The functions `diffQ()` and `diffQ2()` are used to calculate Tm, Tm_1^{D2} and Tm_2^{D2} (Figure 1) and to perform simple graphical operations (e. g., show derivatives). Basically, both functions do not require smoothed data[12] for the MCA. However, it is recommended to use `mcaSmoother()` as a starter function to preprocess (e.g., moderate smoothing, missing value removal, type check) the data automatically. First the approximate Tm, Tm_1^{D2} and Tm_2^{D2} are determined as the `min()` and/or `max()` from the derivatives[13] according to Equation 1, 3 and 4. This approximate peak value is the starting-point for an accurate calculation. The function takes a defined number n (maximum 8) of the left and the right neighbor values and fits a quadratic polynomial[14]. The quadratic regression `lm(Y ~ X I(X^2))` of the X (temperature) against the Y (fluorescence) range gives the coefficients. The optimal quadratic polynomial is chosen based on the highest adjusted R-squared value ($R^2_{adj.}$). In the example of Figure 10 two left and right neighbors were required. The coefficients are used to calculate the root of the quadratic polynomial and thus to determine Tm, Tm_1^{D2} and Tm_2^{D2}. An estimate of a Tm does not neccesarily reflect a valid result. Therefore, several routines were implemented in `diffQ()` and `diffQ2()` which try to catch cases where an experiment went wrong. This includes a test if the data originate from noise, a test which analyses the difference between the approximate Tm and the calculated Tm, and a tests for the goodness of fit value ($R^2_{adj.}$, Normalized-Root-Mean-Squared-Error (NRMSE)) of the quadratic polynomial. The functions will give a warning message and point to the potential error[15]. A good practice is to visualize the output and to control the peak heights (Example in Section "Multiplex analysis of dual melting peaks"). A bimodal probe system (compare Figure 3B) requires the separation of the analysis. The minimum and maximum of the approximate first derivative have to be determined independently. Although `diffQ()` is a major function it has only a simple plot function. By setting the argument `plot = TRUE` plots for single melting curves can be investigated (Figure 9). `diffQ()` accepts further following arguments:

[12]The paramter `rsm` is avilable in both functions to double the temperature resolution. This may also reduce noise.

[13]Besides the algorithm used in `diffQ()` there are further ways to calculate the approximate derivative in R such as `diff()` (**base**) or functions from the **fda** package (Ramsay et al., 2013).

[14]Quadratic polynomials are a good compromise because they are easy to implement, do not tend to swing and fit non-linear progresses sufficiently flexible.

[15]See example for `diffQ()` in Roediger (2013).

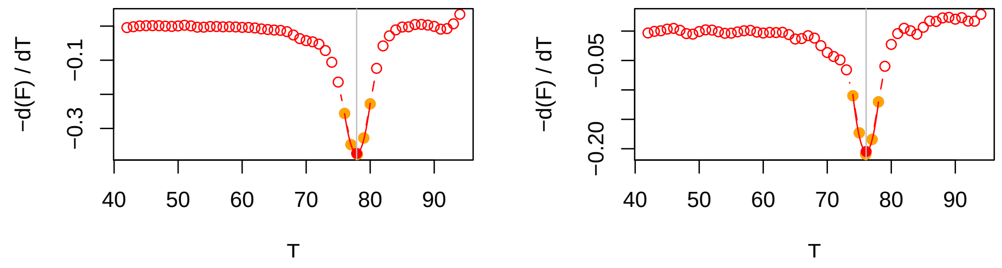


Figure 9: By default `diffQ()` shows no melting peak plot. `diffQ()`, with the argument `plot = TRUE`, can be used to show the melting peak (red dot) and fitted region from the quadratic polynomial (line) of a single melting curve. In this example *HPRT1* (left) and $MLC-2v$ (right) from `MultiMelt` are used.

- `fct` accepts `min` or `max` as argument and is used to define whether to find a local minimum (“negative peak”) or local maximum (“positive peak”).
- `fws` defines the number (n) of left and right neighbors to use for the calculation of the quadratic polynomial.
- `plot` defines if a single plot of the melting peak should be created. If `FALSE` (default) no plot is created.
- `negderiv` is used to change the sign of the derivatives. If `TRUE` (default) then the first negative derivative is calculated. This argument was implemented to compare different quencher / fluorophore combinations (compare Figure 2).
- Functions for a graphical output include `peak` to show the peak values and `deriv` to show the first derivative with the color assigned to `col`. `derivlimits` and `derivlimitsline` show the number of neighbors (n) or the region used to calculate the Tm. `vertiline` draws a vertical line at the Tms (Figure 10).

```
# Load MultiMelt data set.
data(MultiMelt)

# Draw figures on the graphics device in two columns.
par(mfrow = c(1, 2))

# Use mcaSmoother to check and smooth the raw data for HRPT1 (2) and MLC-2v (14)
# with the default setting.
# Plot the first derivative of the two samples.
for (i in c(2, 14)) {
  tmp <- mcaSmoother(MultiMelt[, 1], MultiMelt[, i])
  diffQ(tmp, plot = TRUE, vertiline = TRUE)
}
```

For sophisticated analysis and plots it is recommended to use `diffQ()` as part of the procedure. The arguments (e.g., `peak`, `deriv`) can be used when a plot already exists. `diffQ2()` calls instances of `diffQ()` to calculate Tm_1^{D2} and Tm_2^{D2}. The arguments are similar to `diffQ()`. Both `diffQ()` and `diffQ2()` return objects of the class `list`. Accessing components of lists is done as described elsewhere (Venables et al., 2013; Roediger, 2013) either by name or by number.

Practical applications

Multiplex analysis of single melting peaks

`diffQ2()` was used to investigate effects of the surface capture probe density which is represented by the maximal $refMFI$ value. The Tm, Tm_1^{D2} and Tm_2^{D2} values were determined simultaneously on 12 microbead populations (12-plex) either for *HPRT1* (compare Figure 8) or $MLC-2v$ using data from `MultiMelt`. In the following *HPRT1* was used as example. The corresponding matrix was called `HPRT1`[16].

```
# Load MultiMelt data set.
data(MultiMelt)
```

[16]The script for $MLC-2v$ is similar with the exception that the indices to access the elements of the `data.frame` need to be adapted accordingly.

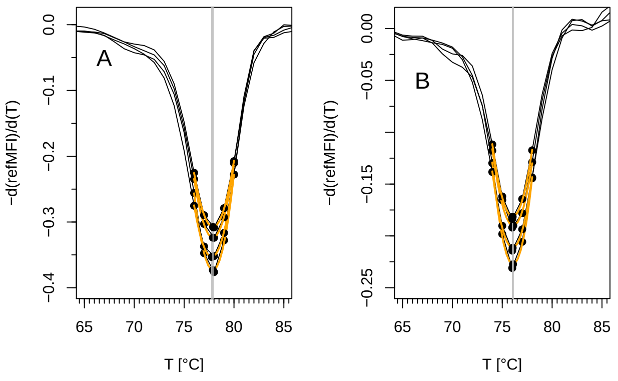


Figure 10: Output of `diffQ()` and fitted region from the quadratic polynomial (orange line) for A) *HPRT1* and B) *MLC−2v* of four randomly selected data from `MultiMelt`.

DP	Tm	Tm_1^{D2}	Tm_2^{D2}
MLC−2v	76.08±0.04	73.99±0.11	78.51±0.13
HPRT1	77.85±0.08	75.39±0.20	80.31±0.14

Table 1: Results of Tm, Tm_1^{D2} and Tm_2^{D2} quantification for *MLC−2v* and *HPRT1*.

```
# Create an empty matrix ("HRPT1") for the diffQ2 results (e.g., Tm).
HPRT1 <- matrix(NA, 12, 4, dimnames = list(colnames(MultiMelt[, 2:13]),
                c("Fluo", "Tm", "Tm1D2", "Tm2D2")))

# Use mcaSmoother to check and smooth the raw data. Apply diffQ2 to the smoothed data,
# calculate the values for the extreme (minimum) and assign the results to "HRPT1".
for (i in 2:13) {
  tmp <- mcaSmoother(MultiMelt[, 1], MultiMelt[, i])
  tmpTM <- diffQ2(tmp, fct = min, verbose = TRUE)
  HPRT1[i-1, 1] <- max(tmp[["y.sp"]])
  HPRT1[i-1, 2] <- as.numeric(tmpTM[["TmD1"]][["Tm"]]) # Tm
  HPRT1[i-1, 3] <- as.numeric(tmpTM[["xTm1.2.D2"]][1]) # Tm1D2
  HPRT1[i-1, 4] <- as.numeric(tmpTM[["xTm1.2.D2"]][2]) # Tm2D2
}
```

The surface capture density was determined by `max(tmp[["y.sp"]])`. Subsequently the data from the matrices `HPRT1` and `MLC2v` were plotted (Figure 11).

```
# Plot the Tm, Tm1D2 and Tm2D2 form the matrix "HRPT1" versus the surface capture
# probe density ("Fluo").
plot(HPRT1[, 1], HPRT1[, 2], xlab = "refMFI", ylab = "T [C]", main = "HPRT1",
     xlim = c(2.1, 2.55), ylim = c(72, 82), pch = 19, col = 1:12, cex = 1.8)

# Add minor tick marks to the abscissa.
require(Hmisc); minor.tick(ny = 10)
points(HPRT1[, 1], HPRT1[, 3], pch = 15)
points(HPRT1[, 1], HPRT1[, 4], pch = 15)

# Add trend lines (lm()) for the peak values.
abline(lm(HPRT1[, 2] ~ HPRT1[, 1])) # Tm
abline(lm(HPRT1[, 3] ~ HPRT1[, 1])) # Tm1D2
abline(lm(HPRT1[, 4] ~ HPRT1[, 1])) # Tm2D2
```

The melting temperature of *MLC−2v* was 76.08±0.04 °C and 77.85±0.08 °C for *HPRT1* on all microbead populations (Table 1). This indicates that the surface capture probe density did not decrease or increase the Tm within the given range. Tm_1^{D2} and Tm_2^{D2} showed a symmetrical pattern with a low variance and therefore support that the start and end of the melting process are similar between the microbead populations. We suggest that the later peak values can be used as an additional means to describe the melting process in more detail.

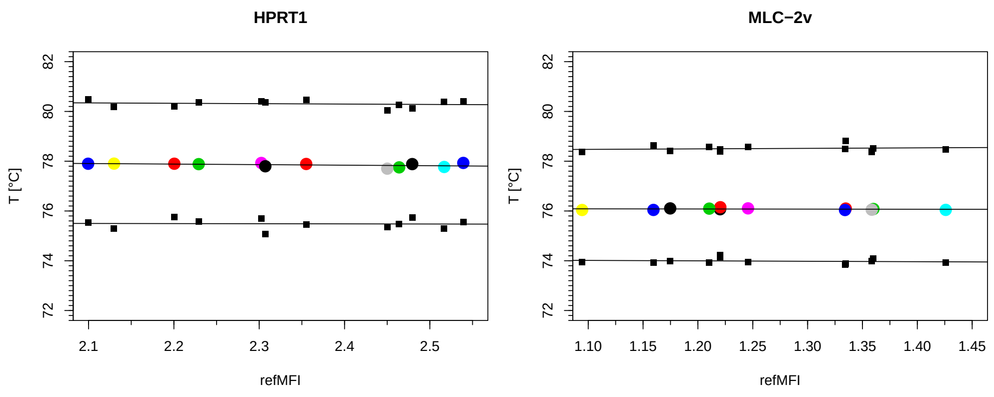


Figure 11: The peak values as function of capture probe density (sorted $refMFI$). Each colored dot represents the Tm and black squares the Tm_1^{D2} and Tm_2^{D2} of $HPRT1$ or $MLC-2v$ on one microbead population.

Multiplex analysis of dual melting peaks

The *bCP*s were hybridized with *DP*s (compare Figure 3) in order to generate bimodal melting peak patterns[17]. Due to the base composition $Poly(dA)_{20}$ *DP*s were expected to melt at lower temperatures than the *DP*s *aCS* and $MLC-2v$. First the temperature and selected probes fluorescence values from the `DMP` data set were arranged in the data frame `data.tmp` and an empty plot was created.

```
# Load DMP data set.
data(DMP)

# Use the temperature (column 1) and fluorescence (column 3, 5, 6), assign them to
# a temporary data frame and add the sample names.
data.tmp <- data.frame(DMP[, 1], DMP[, 3], DMP[, 5], DMP[, 6])
names(data.tmp) <- c("T [C]", "Poly(dA)20 & MLC-2v",
                     "Poly(dA)20 & aCS", "Poly(dA)20")

# Create a plot with the selected raw data.
plot(NA, NA, xlim = c(20, 95), ylim = c(-0.6, 0.6), xlab = "T [C]",
     ylab = "-d(refMFI) / d(T)", main = "", pch = 19, col = 1:12, cex = 1.8)

# Add minor tick marks to the abscissa.
require(Hmisc); minor.tick(nx = 10)
```

Thereafter, the data were preprocessed with `mcaSmoother()` in a loop. The arguments `bg = c(20,35)` and `bgadj` were used to adjust the background signal. This causes `mcaSmoother()` to use the subset of the data between 20 °C and 35 °C for the linear regression and background correction. To determine the Tm of the first probe (positive sign) and the second (negative sign) probe `diffQ()` was used with `min` and `max` for argument `fct`, respectively. In the loop the corresponding Tm values were assigned to the matrix `RES` and the melting curve is drawn. In addition to the lines the Tms were added (Figure 12).

```
# Create an empty matrix ("RES") for the results of the peak values (Tm) and peak
# heights (F).
RES <- matrix(NA, 3, 4, dimnames = list(colnames(data.tmp[, 2:4]),
              c("F 1", "Tm 1", "F 2", "Tm 2")))

# Use mcaSmoother to preprocess the raw data.
# Use a background correction (20-35 degree Celsius).
# Apply the smoothed data to diffQ, calculate the peak values for the extremes
# (minimum and maximum) and assign the results to the matrix "RES".
# Plot the smoothed data with the peak values and peak heights.
for (i in c(1:3)) {
  tmp <- mcaSmoother(data.tmp[, 1], data.tmp[, i + 1], bgadj = TRUE, bg = c(20, 35))
  lines(data.frame(diffQ(tmp, verbose = TRUE)["xy"]), col = i)
  RES[i, 1] <- round(diffQ(tmp, fct = max)[[2]], 2)      #fluoTm
  RES[i, 2] <- round(diffQ(tmp, fct = max)[[1]], 2)      #Tm
```

[17]See Rödiger et al. (2012b) for further details of the probe system.

DP/bCP	F_1	$Tm1$	DP/bCP	F_2	$Tm2$
$Poly(dA)_{20}$	0.59	49.6	$MLC-2v$	-0.57	74.7
–	0.02	91.7	$Poly(dA)_{20}$	-0.08	49.7
$Poly(dA)_{20}$	0.20	47.9	aCS	-0.15	73.9

Table 2: Tm and peak values (F_1, F_2) of the derivatives of bimodal melting curves. The detection probe and microbead bound capture probe pairs (DP/bCP) of $Poly(dA)_{20}/Poly(dT)_{20}$, $MLC-2v/MLC-2v-cap$ and $aCS/aCS-cap$ were analyzed simultaneously.

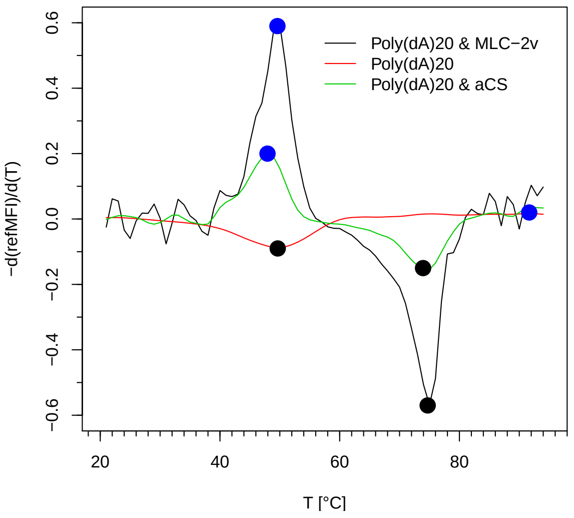


Figure 12: Melting peak pattern of $Poly(dA)_{20}$, $MLC-2v$ and aCS. **Note:** The sign of $Poly(dA)_{20}$ (—) is negative because the *FRET assays* was used (see Figure 2 and Rödiger et al. 2012b for details).

```
  RES[i, 3] <- round(diffQ(tmp, fct = min)[[2]], 2)      #fluoTm
  RES[i, 4] <- round(diffQ(tmp, fct = min)[[1]], 2)      #Tm
}
legend(20, 0.6, names(data.tmp[, 2:4]), lty = 1, bty = "n", col = 1:3)
points(RES[, 2], RES[, 1], pch = 19, col = 4, cex = 2)
points(RES[, 4], RES[, 3], pch = 19, col = 1, cex = 2)
```

Figure 12 shows that $Poly(dA)_{20}$ melts as single sharp peak at circa 48 °C. The detection probes aCS and $MLC-2v$ had a single peak at a higher Tm of circa 74 °C. Both, the number and the separation of the peaks were consistent to the estimated theoretical temperatures[18] (not shown). The results of the matrix RES are shown in Table 2. The Tm of $Poly(dA)_{20}$ alone and $Poly(dA)_{20}$ combined with another detection probe differed slightly. This is presumably due to different capture immobilization strategies and/or *bCP/DP* interactions (unpublished data).

Multiplex SNP detection

SNPs are important diagnostic markers for example in cardiac diseases (Villard et al., 2005; Muthumala et al., 2008). SNPs alter thermodynamic properties of dsDNA and thus the Tm. In proof-of-principle experiments melting curves were obtained from sequences of human *VIM* with a single base exchange[19]. We used the previously reported (Rödiger et al., 2012b) dual-hybridization probe on microbeads to analyze the Tm shift (inset Figure 13).

The temperature resolution was 0.5 °C per step. The DualHyb data were preprocessed with mcaSmoother() in a loop. The Tm and intensity ($fluoTm$) were stored in the matrix RES.

```
# Load DualHyb data set.
data(DualHyb)

# Create an empty matrix ("RES") for the results of the peak values (TmD1)
```

[18]The open source tool *PerlPrimer* by Marshall (2007) was used to estimate the theoretical temperatures.
[19]Cytosine (C) was exchanged to thymine (T) at position 41 of a 60 bp VIM oligonucleotide (see inset of Figure 13).

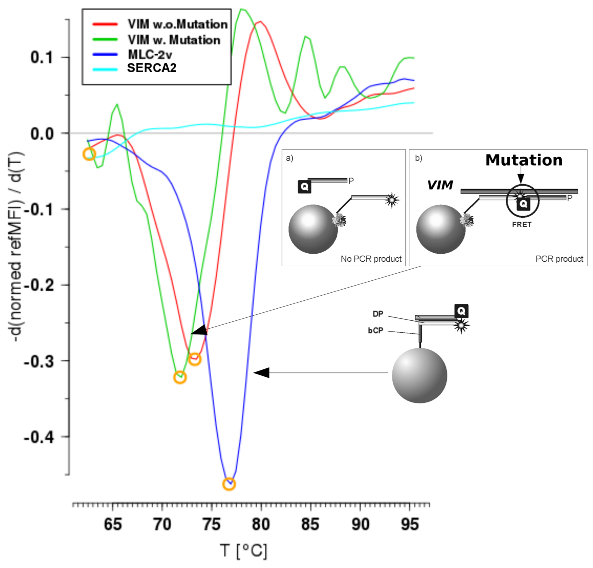


Figure 13: Proof-of-principle SNP detection with dual-hybridization assays on microbead surfaces. The melting temperature of the artificial mutated *VIM* (—) (C → T) is lower compared to the native *VIM* (—). *MLC*−2*v* (—) was used as reference for inter-assay variance. In the negative control *SERCA2* (—) no significant *Tm* was detected.

```
# and calculated peak heights (fluoTm).
RES <- matrix(NA, 4, 2, dimnames = list(colnames(DualHyb[, 2:5]),
              c("fluoTm", "TmD1")))

# Use mcaSmoother to check and smooth the raw data.
# Apply diffQ to the smoothed data, calculate the peak values for the extreme
# (minimum) and assign the results to the matrix "RES".
for (i in c(1:4)) {
  tmp <- mcaSmoother(DualHyb[, 1], DualHyb[, i + 1])
  RES[i, 1] <- round(diffQ(tmp, fct = min)[[2]], 2)     # fluoTm
  RES[i, 2] <- round(diffQ(tmp, fct = min)[[1]], 2)     # Tm
}
```

Calling RES gives the following output:

```
# Call RES to show the peak values and peak heights.
RES
                fluoTm  TmD1
MLC2v            -0.48 76.62
SERCA2           -0.04 62.87
VIM.w.Mutation   -0.33 71.67
VIM.wo.Mutation  -0.32 73.29
```

The algorithm calculated for the muted *VIM* a *Tm* of 71.67 °C which is 1.62 °C lower than the native *VIM* (Figure 13). This is in agreement with expected behavior. The negative control (unspecific *bCP* for *SERCA2*) had a *Tm* of 62.87 °C. But looking at the intensity showed that *SERCA2* is very low (−0.04). For screening purposes a simple cut-off would exclude this sample. The reference *MLC*−2*v* had a *Tm* of 76.62 °C. Thus the method can be applied to identify SNPs. High multiplex level was achieved by using different capture probe microbead combinations. In this example a low temperature resolution of 0.5 °C per step was used. However, since lower heating rates are positively correlated with a higher resolution for SNP analysis generally higher resolutions are recommended. The temperature resolution of the raw data can be analyzed with `diffQ(...,verbose = TRUE)[["temperature"]]`.

Summary and discussion

Experimental hardware platforms often require the development of adapted software, in particular in cases where the hardware affects the signal (e. g., photo bleaching). R is an optimal tool for such scenarios. Within this article we proposed the **MBmca** package for MCA on microbead surfaces. The functions of the package were used for different detection-probe-systems, including direct hybridization of DNA dulexes or dual-hybridization probes for SNP detection. The capture-probe-systems produce unique melting profiles which allowed simple and rapid discrimination of different DNA sequences in one sample. The functions are useful to preprocess and inspect the data and to determine melting temperatures of immobilized DNA fragments. While used in this study for identification and quantification of biomolecules attached to microbeads it is applicable for MCA in solution too (not shown).

It was assumed that a quadratic polynomial at the approximate melting peaks can be used to calculate an accurate Tm. One drawback is that the local quadratic polynomial use a predefined number ($n = 1, \ldots, 8$) of left and right approximate melting peaks neighbors. From experience this approach proved to be reliable for the temperature resolution of $0.5 - -1$ °C per step as used in the present and the study of Rödiger et al. (2012b). Preliminary tests in solution using the LightCycler 2.0 (Roche) with high resolution ($\geq$ 0.1 °C per step) suggest that the method works too (not shown). This implementation is designed to meet the needs of a certain experimental setup and therefore may require evaluation prior to use in new applications. Besides Tm Tm_1^{D2} and Tm_2^{D2} were proposed as additional values to describe early and late phases of the melting process on surfaces. Particularly, in investigations on the impact of the capture probe immobilization strategy or capture probe surface density these values might be useful. Methods to determine the area under the curve (AUC) were not taken into consideration due to the fact that photobleaching and quenching effects play an unknown role and are still an ongoing matter of debate in the literature. In practical terms it is recommended to implement functions from the **MBmca** package in an R GUI (see Valero-Mora and Ledesma 2012), e. g., **RKWard**[20] (Rödiger et al., 2012a). GUIs provide more intuitive means to perform multiplex high-throughput analysis with visual feedback and enrichment with additional information like goodness of fit or peak heights.

Acknowledgment

Part of this work was funded by the BMBF InnoProfile-Projekt 03 IP 611. Grateful thanks belong to the authors of the cited R packages, the R community and the **RKWard** developers.

Bibliography

M. Frasca. *delftfews: delftfews R Extensions*, 2012. URL http://R-Forge.R-project.org/projects/delftfews/. R package version 0.3-163/r164. [p42]

M. B. Gašparič, T. Tengs, J. L. L. Paz, A. Holst-Jensen, M. Pla, T. Esteve, J. Žel, and K. Gruden. Comparison of nine different real-time PCR chemistries for qualitative and quantitative applications in GMO detection. *Analytical and Bioanalytical Chemistry*, 396(6):2023–2029, 2010. URL http://www.ncbi.nlm.nih.gov/pubmed/20087729. [p37]

H. Gudnason, M. Dufva, D. Bang, and A. Wolff. Comparison of multiple DNA dyes for real-time PCR: Effects of dye concentration and sequence composition on DNA amplification and melting temperature. *Nucleic Acids Research*, 35(19):1–8, 2007. URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2095797/. [p39]

C. N. Gundry, J. G. Vandersteen, G. H. Reed, R. J. Pryor, J. Chen, and C. T. Wittwer. Amplicon melting analysis with labeled primers: A closed-tube method for differentiating homozygotes and heterozygotes. *Clinical Chemistry*, 49(3):396–406, 2003. URL http://www.ncbi.nlm.nih.gov/pubmed/12600951. [p37]

F. E. Harrell Jr, C. Dupont, and et al. *Hmisc: Harrell Miscellaneous*, 2013. URL http://CRAN.R-project.org/package=Hmisc. R package version 3.12-2. [p43]

D. C. Ince, L. Hatton, and J. Graham-Cumming. The case for open computer programs. *Nature*, 482 (7386):458–488, 2012. URL http://view.ncbi.nlm.nih.gov/pubmed/22358837. [p39]

O. Marshall. Graphical design of primers with PerlPrimer. *Methods Molecular Biology*, 402:403–414, 2007. URL http://www.ncbi.nlm.nih.gov/pubmed/17951808. [p48]

A. Morin, J. Urban, P. D. Adams, I. Foster, A. Sali, D. Baker, and P. Sliz. Shining light into black boxes. *Science*, 336(6078):159–160, 2012. URL http://www.sciencemag.org/content/336/6078/159. [p39]

[20]Core structures of the **MBmca** were implemented in the **RKWard** melt plug-in (Rödiger et al., 2012a).

A. Muthumala, F. Drenos, P. M. Elliott, and S. E. Humphries. Role of beta adrenergic receptor polymorphisms in heart failure: Systematic review and meta-analysis. *European Journal of Heart Failure*, 10(1):3–13, 2008. URL http://www.ncbi.nlm.nih.gov/pubmed/18158268. [p48]

J. O. Ramsay, H. Wickham, S. Graves, and G. Hooker. *fda: Functional Data Analysis*, 2013. URL http://CRAN.R-project.org/package=fda. R package version 2.4.0. [p44]

K. M. Ririe, R. P. Rasmussen, and C. T. Wittwer. Product differentiation by analysis of DNA melting curves during the polymerase chain reaction. *Analytical Biochemistry*, 245(9):154–160, 1997. URL http://www.ncbi.nlm.nih.gov/pubmed/9056205. [p37]

C. Ritz and A.-N. Spiess. qpcR: An R package for sigmoidal model selection in quantitative real-time polymerase chain reaction analysis. *Bioinformatics*, 24(13):1549–1551, 2008. URL http://bioinformatics.oxfordjournals.org/content/24/13/1549.abstract. [p39]

S. Rödiger, M. Ruhland, C. Schmidt, C. Schröder, K. Grossmann, A. Böhm, J. Nitschke, I. Berger, I. Schimke, and P. Schierack. Fluorescence dye adsorption assay to quantify carboxyl groups on the surface of poly(methyl methacrylate) microbeads. *Analytical Chemistry*, 83(9):3379–3385, 2011. URL www.ncbi.nlm.nih.gov/pubmed/21413805. [p39]

S. Rödiger, T. Friedrichsmeier, P. Kapat, and M. Michalke. RKWard: A comprehensive graphical user interface and integrated development environment for statistical analysis with R. *Journal of Statistical Software*, 49(9):1–34, 2012a. URL http://www.jstatsoft.org/v49/i09. [p37, 50]

S. Rödiger, P. Schierack, A. Böhm, J. Nitschke, I. Berger, U. Frömmel, C. Schmidt, M. Ruhland, I. Schimke, D. Roggenbuck, W. Lehmann, and C. Schröder. A highly versatile microscope imaging technology platform for the multiplex real-time detection of biomolecules and autoimmune antibodies. In *Molecular Diagnostics*, volume 133 of *Advances in Biochemical Bioengineering/Biotechnology*, pages 35–74. 2012b. doi: 10.1007/10_2011_132. [p37, 38, 39, 40, 47, 48, 50]

S. Roediger. *MBmca: Nucleic Acid Melting Curve Analysis on Microbead Surfaces with R*, 2013. URL http://CRAN.R-project.org/package=MBmca. R package version 0.0.2-1. [p40, 43, 44, 45]

P. Rousseeuw, C. Croux, V. Todorov, A. Ruckstuhl, M. Salibian-Barrera, T. Verbeke, M. Koller, and M. Maechler. *robustbase: Basic Robust Statistics*, 2013. URL http://CRAN.R-project.org/package=robustbase. R package version 0.9-10. [p40]

M. M. Sekar, W. Bloch, and P. M. St John. Comparative study of sequence-dependent hybridization kinetics in solution and on microspheres. *Nucleic Acids Research*, 33(1):366–375, 2005. URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC546151/. [p37]

The signal Developers. *signal: Signal Processing*, 2013. URL http://R-Forge.R-project.org/projects/signal/. [p42]

P. M. Valero-Mora and R. Ledesma. Graphical user interfaces for R. *Journal of Statistical Software*, 49(1):1–8, 2012. URL http://www.jstatsoft.org/v49/i01/paper. [p50]

W. N. Venables, D. M. Smith, and the R Core Team. *An Introduction to R*. R Foundation for Statistical Computing, Vienna, Austria, 2013. URL http://www.CRAN.R-project.org/doc/manuals/R-intro.pdf. [p40, 45]

E. Villard, L. Duboscq-Bidot, P. Charron, A. Benaiche, V. Conraads, N. Sylvius, and M. Komajda. Mutation screening in dilated cardiomyopathy: Prominent role of the beta myosin heavy chain gene. *European Heart Journal*, 26(8):794–803, 2005. URL http://www.ncbi.nlm.nih.gov/pubmed/15769782. [p48]

A. Willitzki, R. Hiemann, V. Peters, U. Sack, P. Schierack, S. Rödiger, U. Anderer, K. Conrad, D. P. Bogdanos, D. Reinhold, and D. Roggenbuck. New platform technology for comprehensive serological diagnostics of autoimmune diseases. *Clinical & Developmental Immunology*, 2012:1–8, 2012. URL https://www.ncbi.nlm.nih.gov/pubmed/23316252. [p38]

A. Zeileis and G. Grothendieck. zoo: S3 infrastructure for regular and irregular time series. *Journal of Statistical Software*, 14(6):1–27, 2005. URL http://www.jstatsoft.org/v14/i06. [p42]

L. Zhou, L. Wang, R. Palais, R. Pryor, and C. T. Wittwer. High-resolution DNA melting analysis for simultaneous mutation scanning and genotyping in solution. *Clinical Chemistry*, 51(10):1770–1777, 2005. URL http://www.ncbi.nlm.nih.gov/pubmed/16189378. [p37]

Stefan Rödiger
Charité-Universitätsmedizin Berlin
and
Lausitz University of Applied Sciences
Faculty II
Germany
`Stefan.Roediger@HS-Lausitz.de`

Alexander Böhm
Lausitz University of Applied Sciences
Faculty I
Germany
`Alexander.Boehm@HS-Lausitz.de`

Ingolf Schimke
Charité-Universitätsmedizin Berlin
Germany
`ingolf.schimke@charite.de`

Performance Attribution for Equity Portfolios

by Yang Lu and David Kane

Abstract The **pa** package provides tools for conducting performance attribution for long-only, single currency equity portfolios. The package uses two methods: the Brinson-Hood-Beebower model (hereafter referred to as the Brinson model) and a regression-based analysis. The Brinson model takes an ANOVA-type approach and decomposes the active return of any portfolio into asset allocation, stock selection, and interaction effect. The regression-based analysis utilizes estimated coefficients, based on a regression model, to attribute active return to different factors.

Introduction

Almost all portfolio managers measure performance with reference to a benchmark. The difference in return between a portfolio and the benchmark is its active return. Performance attribution decomposes the active return. The two most common approaches are the Brinson model (Brinson et al., 1986) and a regression-based analysis (Grinold, 2006).

Portfolio managers use different variations of the two models to assess the performance of their portfolios. Managers of fixed income portfolios include yield-curve movements in the model (Lord, 1997) while equity managers who focus on the effect of currency movements use variations of the Brinson model to incorporate "local risk premium" (Singer and Karnosky, 1995). In contrast, in this paper we focus on attribution models for long-only equity portfolios without considering any currency effect.[1]

The **pa** package provides tools for conducting both methods for long-only, single currency equity portfolios.[2] The Brinson model takes an ANOVA-type approach and decomposes the active return of any portfolio into asset allocation, stock selection, and interaction effects. The regression-based analysis utilizes estimated coefficients from a linear model to estimate the contributions from different factors.

Data

We demonstrate the use of the **pa** package with a series of examples based on data from MSCI Barra's Global Equity Model II (GEM2).[3] The original data set contains selected attributes such as industry, size, country, and various style factors for a universe of approximately 48,000 securities on a monthly basis. For illustrative purposes, this article uses three modified versions of the original data set (`year`, `quarter`, and `jan`), each containing 3000 securities. The data frame, `quarter`, is a subset of `year`, containing the data of the first quarter. The data frame, `jan`, is a subset of `quarter` with the data from January, 2010.

```
> data(year)
> names(year)
 [1] "barrid"    "name"      "return"    "date"      "sector"    "momentum"
 [7] "value"     "size"      "growth"    "cap.usd"   "yield"     "country"
[13] "currency"  "portfolio" "benchmark"
```

See `?year` for information on the different variables. The top 200 securities, based on `value` scores, in January are selected as portfolio holdings and are held through December 2010 with monthly rebalances to maintain equal-weighting. The benchmark for this portfolio is defined as the largest 1000 securities based on `size` each month. The benchmark is cap-weighted.

Here is a sample of rows and columns from the data frame `year`:

[1]See Morningstar Inc (2009) for a comprehensive discussion of conventional attribution methods.

[2]There are several R packages which provide related functionality: **portfolio** (Enos and Kane, 2006) enables users to analyze and implement equity portfolios; **PerformanceAnalytics** (Carl and Peterson, 2013) provides a collection of econometric functions for performance and risk analysis; the Rmetrics suite contains a collection of functions for computational finance (Rmetrics Association, 2013). Although **PerformanceAnalytics** and the Rmetrics suite provide a variety of tools, they do not provide for the attribution of returns using the Brinson Model.

[3]See `www.msci.com` and Menchero et al. (2008) for more information.

```
                              name   return       date     sector  size
44557  BLUE STAR OPPORTUNITIES CORP  0.00000 2010-01-01     Energy  0.00
25345  SEADRILL                     -0.07905 2010-01-01     Energy -0.27
264017 BUXLY PAINTS (PKR10)         -0.01754 2010-05-01  Materials  0.00
380927 CDN IMPERIAL BK OF COMMERCE   0.02613 2010-08-01 Financials  0.52
388340 CDN IMPERIAL BK OF COMMERCE  -0.00079 2010-11-01 Financials  0.55
       country portfolio benchmark
44557      USA     0.000  0.000000
25345      NOR     0.000  0.000427
264017     PAK     0.005  0.000000
380927     CAN     0.005  0.000012
388340     CAN     0.005  0.000012
```

The portfolio has 200 equal-weighted holdings each month. The row for Canadian Imperial Bank of Commerce indicates that it is one of the 200 portfolio holdings with a weight of 0.5% in 2010. Its return was 2.61% in August, and close to flat in November.

Brinson model

Consider an equity portfolio manager who uses the S&P 500 as the benchmark. In a given month, she outperformed the S&P by 3%. Part of that performance was due to the fact that she allocated more weight of the portfolio to certain sectors that performed well. Call this the *allocation effect*. Part of her outperformance was due to the fact that some of the stocks she selected did better than their sector as a whole. Call this the *selection effect*. The residual can then be attributed to an interaction between allocation and selection – the *interaction effect*. The Brinson model provides mathematical definitions for these terms and methods for calculating them.

The example above uses sector as the classification scheme when calculating the allocation effect. But the same approach can work with any other variable which places each security into one, and only one, discrete category: country, industry, and so on. In fact, a similar approach can work with continuous variables that are split into discrete ranges: the highest quintile of market cap, the second highest quintile and so forth. For generality, we will use the term "category" to describe any classification scheme which places each security in one, and only one, category.

Notations:

- w_i^B is the weight of security i in the benchmark.
- w_i^P is the weight of security i in the portfolio.
- W_j^B is the weight of category j in the benchmark. $W_j^B = \sum w_i^B, i \in j$.
- W_j^P is the weight of a category j in the portfolio. $W_j^P = \sum w_i^P, i \in j$.
- The sum of the weight w_i^B, w_i^P, W_j^B, and W_j^P is 1, respectively.
- r_i is the return of security i.
- R_j^B is the return of a category j in the benchmark. $R_j^B = \sum w_i^B r_i, i \in j$.
- R_j^P is the return of a category j in the portfolio. $R_j^P = \sum w_i^P r_i, i \in j$.

The return of a portfolio, R_P, can be calculated in two ways:

- On an individual security level by summing over n stocks: $R_P = \sum\limits_{i=1}^{n} w_i^P r_i$.
- On a category level by summing over N categories: $R_P = \sum\limits_{j=1}^{N} W_j^P R_j^P$.

Similar definitions apply to the return of the benchmark, R_B,

$$R_B = \sum_{i=1}^{n} w_i^B r_i = \sum_{j=1}^{N} W_j^B R_j^B.$$

Active return of a portfolio, R_{active}, is a performance measure of a portfolio relative to its benchmark. The two conventional measures of active return are arithmetic and geometric. The **pa** package implements the arithmetic measure of the active return for a single-period Brinson model because an arithmetic difference is more intuitive than a ratio over a single period.

The arithmetic active return of a portfolio, R_{active}, is the portfolio return R_P less the benchmark return R_B:

$$R_{active} = R_P - R_B.$$

Since the category weights of the portfolio are generally different from those of the benchmark, allocation plays a role in the active return, R_{active}. The same applies to stock selection effects. Within a given category, the portfolio and the benchmark will rarely have exactly the same holdings. Allocation effect $R_{allocation}$ and selection effect $R_{selection}$ over N categories are defined as:

$$R_{allocation} = \sum_{j=1}^{N} \left(W_j^P - W_j^B\right) R_j^B,$$

and

$$R_{selection} = \sum_{j=1}^{N} W_j^B \left(R_j^P - R_j^B\right).$$

The intuition behind the allocation effect is that a portfolio would produce different returns with different allocation schemes (W_j^P vs. W_j^B) while having the same stock selection and thus the same return (R_j^B) for each category. The difference between the two returns, caused by the allocation scheme, is called the allocation effect ($R_{allocation}$). Similarly, two different returns can be produced when two portfolios have the same allocation (W_j^B) yet dissimilar returns due to differences in stock selection within each category (R_j^p vs. R_j^B). This difference is the selection effect ($R_{selection}$).

Interaction effect ($R_{interaction}$) is the result of subtracting return due to allocation $R_{allocation}$ and return due to selection $R_{selection}$ from the active return R_{active}:

$$R_{interaction} = R_{active} - R_{allocation} - R_{selection}.$$

The Brinson model allows portfolio managers to analyze the active return of a portfolio using any attribute of a security, such as country or sector. Unfortunately, it is very hard to expand the analysis beyond two categories. As the number of categories increases, the number of terms to be included in the Brinson model grows exponentially; this procedure is thus subject to the curse of dimensionality. To some extent, the regression-based model detailed later ameliorates this problem.

Brinson tools

Brinson analysis is run by calling the function `brinson` to produce an object of class "brinson".

```
> data(jan)
> br.single <- brinson(x = jan, date.var = "date", cat.var = "sector",
+             bench.weight = "benchmark", portfolio.weight = "portfolio",
+             ret.var = "return")
```

The data frame, `jan`, contains all the information necessary to conduct a single-period Brinson analysis. `date.var`, `cat.var`, and `return` identify the columns containing the date, the factor to be analyzed, and the return variable, respectively. `bench.weight` and `portfolio.weight` specify the name of the benchmark weight column and that of the portfolio weight column in the data frame.

Calling `summary` on the resulting object `br.single` of class "brinson" reports essential information about the input portfolio (including the number of securities in the portfolio and the benchmark as well as sector exposures) and the results of the Brinson analysis (both by sector and aggregate).

```
> summary(br.single)
Period:                            2010-01-01
Methodology:                       Brinson
Securities in the portfolio:       200
Securities in the benchmark:       1000

Exposures
            Portfolio Benchmark     Diff
Energy          0.085    0.2782 -0.19319
Materials       0.070    0.0277  0.04230
Industrials     0.045    0.0330  0.01201
ConDiscre       0.050    0.0188  0.03124
ConStaples      0.030    0.0148  0.01518
HealthCare      0.015    0.0608 -0.04576
```

```
Financials      0.370    0.2979  0.07215
InfoTech        0.005    0.0129 -0.00787
TeleSvcs        0.300    0.1921  0.10792
Utilities       0.030    0.0640 -0.03399

Returns
$'Attribution by category in bps'
            Allocation Selection Interaction
Energy         110.934    -37.52      26.059
Materials      -41.534      0.48       0.734
Industrials      0.361      1.30       0.473
ConDiscre      -28.688     -4.23      -7.044
ConStaples       5.467     -3.59      -3.673
HealthCare      -6.692     -4.07       3.063
Financials     -43.998     70.13      16.988
InfoTech        -3.255     -5.32       3.255
TeleSvcs       -23.106     41.55      23.348
Utilities       16.544     83.03     -44.108
Total          -13.966    141.77      19.095

$Aggregate
                   2010-01-01
Allocation Effect    -0.00140
Selection Effect      0.01418
Interaction Effect    0.00191
Active Return         0.01469
```

The `br.single` summary shows that the active return of the portfolio, in January, 2010 was 1.47%. This return can be decomposed into allocation effect (-0.14%), selection effect (1.42%), and interaction effect (0.19%).

```
> plot(br.single, var = "sector", type = "return")
```

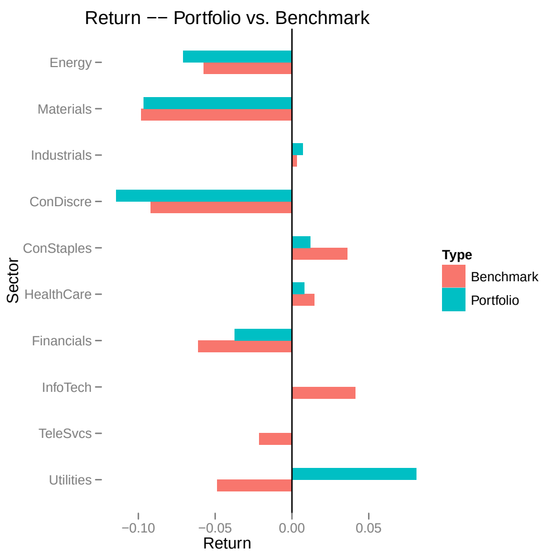


Figure 1: Sector Return.

Figure 1 is a visual representation of the return of both the portfolio and the benchmark sector by sector in January, 2010. Utilities was the sector with the highest active return in the portfolio.

To obtain Brinson attribution on a multi-period data set, one calculates allocation, selection and interaction within each period and aggregates them across time. There are three methods for this – arithmetic, geometric, and optimized linking (Menchero, 2000). The arithmetic attribution model calculates active return and contributions due to allocation, selection, and interaction in each period and sums them over multiple periods.

In practice, analyzing a single-period portfolio is meaningless as portfolio managers and their clients are more interested in the performance of a portfolio over multiple periods. To apply the

Brinson model over time, we can use the function brinson and input a multi-period data set (for instance, quarter) as shown below.

```
> data(quarter)
> br.multi <- brinson(quarter, date.var = "date", cat.var = "sector",
+           bench.weight = "benchmark", portfolio.weight = "portfolio",
+           ret.var = "return")
```

The object br.multi of class "brinsonMulti" is an example of a multi-period Brinson analysis.

```
> exposure(br.multi, var = "size")
$Portfolio
     2010-01-01 2010-02-01 2010-03-01
Low       0.140      0.140      0.155
2         0.050      0.070      0.045
3         0.175      0.145      0.155
4         0.235      0.245      0.240
High      0.400      0.400      0.405

$Benchmark
     2010-01-01 2010-02-01 2010-03-01
Low      0.0681     0.0568     0.0628
2        0.0122     0.0225     0.0170
3        0.1260     0.1375     0.1140
4        0.2520     0.2457     0.2506
High     0.5417     0.5374     0.5557

$Diff
     2010-01-01 2010-02-01 2010-03-01
Low      0.0719   0.083157     0.0922
2        0.0378   0.047456     0.0280
3        0.0490   0.007490     0.0410
4       -0.0170  -0.000719    -0.0106
High    -0.1417  -0.137385    -0.1507
```

The exposure method on the br.multi object shows the exposure of the portfolio and the benchmark, and their difference based on a user-specified variable. Here, it shows the exposure on size. We can see that the portfolio overweights the benchmark in the lowest quintile in size and underweights in the highest quintile.

```
> returns(br.multi, type = "arithmetic")
$Raw
             2010-01-01 2010-02-01 2010-03-01
Allocation      -0.0014     0.0062     0.0047
Selection        0.0142     0.0173    -0.0154
Interaction      0.0019    -0.0072    -0.0089
Active Return    0.0147     0.0163    -0.0196

$Aggregate
             2010-01-01, 2010-03-01
Allocation                   0.0095
Selection                    0.0160
Interaction                 -0.0142
Active Return                0.0114
```

The returns method shows the results of the Brinson analysis applied to the data from January, 2010 through March, 2010. The first portion of the returns output shows the Brinson attribution in individual periods. The second portion shows the aggregate attribution results. The portfolio formed by top 200 value securities in January had an active return of 1.14% over the first quarter of 2010. The allocation and the selection effects contributed 0.95% and 1.6% respectively; the interaction effect decreased returns by 1.42%.

Regression

One advantage of a regression-based approach is that such analysis allows one to define their own attribution model by easily incorporating multiple variables in the regression formula. These variables can be either discrete or continuous.

Suppose a portfolio manager wants to find out how much each of the value, growth, and momentum scores of her holdings contributes to the overall performance of the portfolio. Consider the following linear regression without the intercept term based on a single-period portfolio of n securities with k different variables:

$$\mathbf{r}_n = \mathbf{X}_{n,k}\mathbf{f}_k + \mathbf{u}_n$$

where

- $\mathbf{r}_n$ is a column vector of length n. Each element in $\mathbf{r}_n$ represents the return of a security in the portfolio.
- $\mathbf{X}_{n,k}$ is an n by k matrix. Each row represents k attributes of a security. There are n securities in the portfolio.
- $\mathbf{f}_k$ is a column vector of length k. The elements are the estimated coefficients from the regression. Each element represents the *factor return* of an attribute.
- $\mathbf{u}_n$ is a column vector of length n with residuals from the regression.

In the case of this portfolio manager, suppose that she only has three holdings in her portfolio. r_3 is thus a 3 by 1 matrix with returns of all her three holdings. The matrix $\mathbf{X}_{3,3}$ records the score for each of the three factors (value, growth, and momentum) in each row. $\mathbf{f}_3$ contains the estimated coefficients of a regression $\mathbf{r}_3$ on $\mathbf{X}_{3,3}$.

The active exposure of each of the k variables, X_i, $i \in k$, is expressed as

$$X_i = \mathbf{w}_{active}{'}\mathbf{x}_{n,i},$$

where X_i is the value representing the active exposure of the attribute i in the portfolio, $\mathbf{w}_{active}$ is a column vector of length n containing the active weight of every security in the portfolio, and $\mathbf{x}_{n,i}$ is a column vector of length n with attribute i for all securities in the portfolio. Active weight of a security is defined as the difference between the portfolio weight of the security and its benchmark weight.

Using the example mentioned above, the active exposure of the attribute `value`, X_{value} is the product of $\mathbf{w}_{active}{'}$ (containing active weight of each of the three holdings) and $\mathbf{x}_3$ (containing value scores of the three holdings).

The contribution of a variable i, R_i, is thus the product of the factor returns for the variable i, f_i and the active exposure of the variable i, X_i. That is,

$$R_i = f_i X_i.$$

Continuing the example, the contribution of value is the product of f_{value} (the estimated coefficient for value from the linear regression) and X_{value} (the active exposure of value as shown above).

Therefore, the active return of the portfolio R_{active} is the sum of contributions of all k variables and the residual u (a.k.a. the interaction effect),

$$R_{active} = \sum_{i=1}^{k} R_i + u.$$

For instance, a hypothetical portfolio has three holdings (A, B, and C), each of which has two attributes – size and value.

```
  Return Name Size Value Active_Weight
1    0.3    A  1.2   3.0           0.5
2    0.4    B  2.0   2.0           0.1
3    0.5    C  0.8   1.5          -0.6
```

Following the procedure as mentioned, the factor returns for size and value are -0.0313 and -0.1250. The active exposure of size is 0.32 and that of value is 0.80. The active return of the portfolio is -11% which can be decomposed into the contribution of size and that of value based on the regression model. Size contributes 1% of the negative active return of the portfolio and value causes the portfolio to lose the other 10.0%.

Regression tools

The **pa** package provides tools to analyze both single-period and multi-period data frames.

```
> rb.single <- regress(jan, date.var = "date", ret.var = "return",
+           reg.var = c("sector", "growth", "size"),
+           benchmark.weight = "benchmark", portfolio.weight = "portfolio")
```

reg.var specifies the columns containing variables whose contributions are to be analyzed. Each of the reg.var input variables corresponds to a particular column in $\mathbf{X}_{n,k}$ from the aforementioned regression model. ret.var specifies the column in the data frame jan based on which $\mathbf{r}_n$ in the regression model is formed.

```
> exposure(rb.single, var = "growth")
     Portfolio Benchmark    Diff
Low      0.305    0.2032  0.1018
2        0.395    0.4225 -0.0275
3        0.095    0.1297 -0.0347
4        0.075    0.1664 -0.0914
High     0.130    0.0783  0.0517
```

Calling exposure with a specified var yields information on the exposure of both the portfolio and the benchmark by that variable. If var is a continuous variable, for instance, growth, the exposure will be shown in 5 quantiles. Majority of the high value securities in the portfolio in January have relatively low growth scores.

```
> summary(rb.single)
Period:                            2010-01-01
Methodology:                       Regression
Securities in the portfolio:       200
Securities in the benchmark:       1000

Returns
                 2010-01-01
sector             0.003189
growth             0.000504
size               0.002905
Residual           0.008092
Portfolio Return  -0.029064
Benchmark Return  -0.043753
Active Return      0.014689
```

The summary method shows the number of securities in the portfolio and the benchmark, and the contribution of each input variable according to the regression-based analysis. In this case, the portfolio made a loss of 2.91% and the benchmark lost 4.38%. Therefore, the portfolio outperformed the benchmark by 1.47%. sector, growth, and size contributed 0.32%, 0.05%, and 0.29%, respectively.

Regression-based analysis can be applied to a multi-period data frame by calling the same method regress. By typing the name of the object rb.multi directly, a short summary of the analysis is provided, showing the starting and ending period of the analysis, the methodology, and the average number of securities in both the portfolio and the benchmark.

```
> rb.multi <- regress(year, date.var = "date", ret.var = "return",
+          reg.var = c("sector", "growth", "size"),
+          benchmark.weight = "benchmark", portfolio.weight = "portfolio")
> rb.multi
Period starts:                     2010-01-01
Period ends:                       2010-12-01
Methodology:                       Regression
Securities in the portfolio:       200
Securities in the benchmark:       1000
```

The regression-based summary shows that the contribution of each input variable in addition to the basic information on the portfolio. The summary suggests that the active return of the portfolio in year 2010 is 10.1%. The Residual number indicates the contribution of the interaction among various variables including sector, growth, and size. Based on the regression model, size contributed to the lion's share of the active return.

```
> summary(rb.multi)
Period starts:                     2010-01-01
Period ends:                       2010-12-01
Methodology:                       Regression
```

```
Avg securities in the portfolio:   200
Avg securities in the benchmark:   1000

Returns
$Raw
                 2010-01-01 2010-02-01 2010-03-01
sector               0.0032     0.0031     0.0002
growth               0.0005     0.0009    -0.0001
size                 0.0029     0.0295     0.0105
Residual             0.0081    -0.0172    -0.0302
Portfolio Return    -0.0291     0.0192     0.0298
Benchmark Return    -0.0438     0.0029     0.0494
Active Return        0.0147     0.0163    -0.0196
                 2010-04-01 2010-05-01 2010-06-01
sector               0.0016     0.0039     0.0070
growth               0.0001     0.0002     0.0004
size                 0.0135     0.0037     0.0018
Residual            -0.0040     0.0310     0.0183
Portfolio Return    -0.0080    -0.0381     0.0010
Benchmark Return    -0.0192    -0.0769    -0.0266
Active Return        0.0113     0.0388     0.0276
                 2010-07-01 2010-08-01 2010-09-01
sector               0.0016     0.0047    -0.0022
growth              -0.0005     0.0005    -0.0006
size                 0.0064     0.0000     0.0096
Residual            -0.0324     0.0173    -0.0220
Portfolio Return     0.0515    -0.0119     0.0393
Benchmark Return     0.0764    -0.0344     0.0545
Active Return       -0.0249     0.0225    -0.0152
                 2010-10-01 2010-11-01 2010-12-01
sector               0.0015    -0.0044    -0.0082
growth              -0.0010    -0.0004     0.0010
size                 0.0022     0.0130     0.0056
Residual             0.0137     0.0175    -0.0247
Portfolio Return     0.0414    -0.0036     0.0260
Benchmark Return     0.0249    -0.0293     0.0523
Active Return        0.0165     0.0257    -0.0263

$Aggregate
                 2010-01-01, 2010-12-01
sector                           0.0120
growth                           0.0011
size                             0.1030
Residual                        -0.0269
Portfolio Return                 0.1191
Benchmark Return                 0.0176
Active Return                    0.1015
```

Figure 2 displays both the cumulative portfolio and benchmark returns from January, 2010 through December, 2010. It suggests that the portfolio, consisted of high `value` securities in January, consistently outperformed the benchmark in 2010. Outperformance in May and June helped the overall positive active return in 2010 to a large extent.

```
> plot(rb.multi, var = "sector", type = "return")
```

Conclusion

In this paper, we describe two widely-used methods for performance attribution – the Brinson model and the regression-based approach, and provide a simple collection of tools to implement these two methods in R with the **pa** package. A comprehensive package, **portfolio** (Enos and Kane, 2006), provides facilities to calculate exposures and returns for equity portfolios. It is possible to use the **pa** package based on the output from the **portfolio** package. Further, the flexibility of R itself allows users to extend and modify these packages to suit their own needs and/or execute their preferred attribution

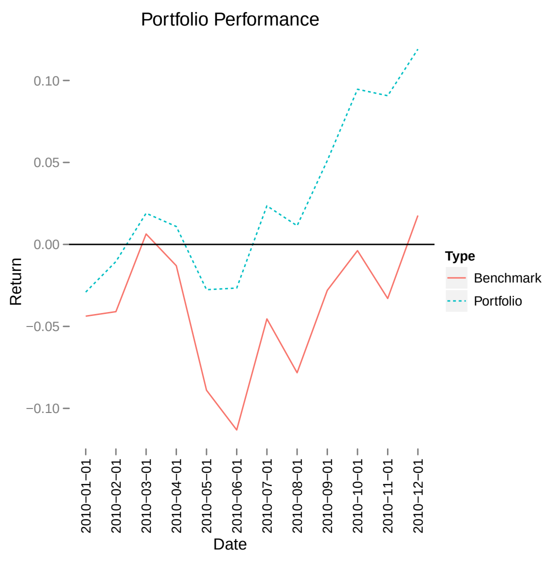


Figure 2: Performance Attribution.

methodology. Before reaching that level of complexity, however, **pa** provides a good starting point for basic performance attribution.

Bibliography

G. Brinson, R. Hood, and G. Beebower. Determinants of portfolio performance. *Financial Analysts Journal*, 42(4):39–44, Jul.–Aug. 1986. URL http://www.jstor.org/stable/4478947. [p53]

P. Carl and B. G. Peterson. *PerformanceAnalytics: Econometric tools for performance and risk analysis*, 2013. URL http://CRAN.R-project.org/package=PerformanceAnalytics. R package version 1.1.0. [p53]

J. Enos and D. Kane. Analysing equity portfolios in R. *R News*, 6(2):13–19, MAY 2006. URL http://CRAN.R-project.org/doc/Rnews. [p53, 60]

R. Grinold. Attribution. Modeling asset characteristics as portfolios. *The Journal of Portfolio Management*, page 19, Winter 2006. [p53]

T. Lord. The attribution of portfolio and index returns in fixed income. *Journal of Performance Measurement*, Fall:45–57, 1997. [p53]

J. Menchero. An optimized approach to linking attribution effects over time. *Journal of Performance Measurement*, 5(1):36–42, 2000. [p56]

J. Menchero, A. Morozov, and P. Shepard. The Barra Global Equity Model (GEM2). *MSCI Barra Research Notes*, Sept. 2008. [p53]

Morningstar Inc. Equity performance attribution methodology. *Morningstar Methodology Paper*, March. 2009. URL http://corporate.morningstar.com/us/documents/MethodologyDocuments/MethodologyPapers/EquityPerformanceAttributionMeth.pdf. [p53]

Rmetrics Association. Rmetrics, 2013. URL https://www.rmetrics.org/. [p53]

B. Singer and D. Karnosky. The general framework for global investment management and performance attribution. *The Journal of Portfolio Management*, Winter:84–92, 1995. [p53]

Yang Lu
Williams College
2896 Paresky Center
Williamstown, MA 01267
United States
yang.lu2014@gmail.com

David Kane
Harvard University
IQSS
1737 Cambridge Street
CGIS Knafel Building, Room 350
Cambridge, MA 02138
United States
dave.kane@gmail.com

ExactCIdiff: An R Package for Computing Exact Confidence Intervals for the Difference of Two Proportions

by Guogen Shan and Weizhen Wang

Abstract Comparing two proportions through the difference is a basic problem in statistics and has applications in many fields. More than twenty confidence intervals (Newcombe, 1998a,b) have been proposed. Most of them are approximate intervals with an asymptotic infimum coverage probability much less than the nominal level. In addition, large sample may be costly in practice. So exact optimal confidence intervals become critical for drawing valid statistical inference with accuracy and precision. Recently, Wang (2010, 2012) derived the exact smallest (optimal) one-sided $1-\alpha$ confidence intervals for the difference of two paired or independent proportions. His intervals, however, are computer-intensive by nature. In this article, we provide an R package **ExactCIdiff** to implement the intervals when the sample size is not large. This would be the first available package in R to calculate the exact confidence intervals for the difference of proportions. Exact two-sided $1-\alpha$ interval can be easily obtained by taking the intersection of the lower and upper one-sided $1-\alpha/2$ intervals. Readers may jump to Examples 1 and 2 to obtain these intervals.

Introduction

The comparison of two proportions through the difference is one of the basic statistical problems. One-sided confidence intervals are of interest if the goal of a study is to show superiority (or inferiority), e.g., that a treatment is better than the control. If both limits are of interest, then two-sided intervals are needed.

In practice, most available intervals, see Newcombe (1998a,b), are approximate ones, i.e., the probability that the interval includes the difference of two proportions, the so-called coverage probability, is not always at least the nominal level although the interval aims at it. Also, even with a large sample size, the infimum coverage probability may still be much less than the nominal level and does not converge to this quantity. In fact, the Wald type interval has an infimum coverage probability zero for any sample sizes and any nominal level $1-\alpha$ even though it is based on asymptotic normality, as pointed out by Agresti and Coull (1998) and Brown et al. (2001). See Wang and Zhang (in press) for more examples. Therefore, people may question of using large samples when such approximate intervals are employed since they cannot guarantee a correct coverage.

Exact intervals which assure an infimum coverage probability of at least $1-\alpha$ do not have this problem. But they are typically computer-intensive by nature. In this paper, a new R package **ExactCIdiff** (Shan and Wang, 2013) is presented which implements the computation of such intervals as proposed in Wang (2010, 2012). The package is available from CRAN at `http://CRAN.R-project.org/package=ExactCIdiff/`. This package contains two main functions: `PairedCI()` and `BinomCI()`, where `PairedCI()` is for calculating lower one-sided, upper one-sided and two-sided confidence intervals for the difference of two paired proportions and `BinomCI()` is for the difference of two independent proportions when the sample size is small to medium. Results from **ExactCIdiff** are compared with those from the function `ci.pd()` in the R package **Epi** (Carstensen et al., 2013), and the `PROC FREQ` procedure in the software SAS (SAS Institute Inc., 2011).

Depending on how the data are collected, one group of three intervals is needed for the difference of two paired proportions and another group for the difference of two independent proportions. Pointed out by Mehrotra et al. (2003), an exact inference procedure may result in poor powerful analysis if an impropriate statistic is employed. Wang's one-sided intervals (Wang, 2010, 2012), obtained through a carefully inductive construction on an order, are optimal in the sense that they are a subset of any other one-sided $1-\alpha$ intervals that preserve the same order, and are called the smallest intervals. See more details in the paragraph following (6). From the mathematical point of view, his intervals are not nested, see Lloyd and Kabaila (2010); on the other hand, for three commonly used confidence levels, 0.99, 0.95, 0.9, the intervals are nested based on our numerical study.

Although R provides exact confidence intervals for one proportion, e.g., the function `exactci()` in the package **PropCIs** (Scherer, 2013), the function `binom.exact()` in the package **exactci** (Fay, 2010, 2012) and the function `binom.test()` in the package **stats** (Version 2.15.2), there is no exact confidence interval available in R, to the best of our knowledge, for the difference of two proportions, which is widely used in practice. **ExactCIdiff** is the first available R package to serve this purpose. The R package **ExactNumCI** (Sun and Park, 2013) claims that its function `pdiffCI()` generates an exact

	S_2	F_2	
S_1	N_{11}, p_{11}	N_{12}, p_{12}	$p_1 = p_{11} + p_{12}$
F_1	N_{21}, p_{21}	N_{22}, p_{22}	
	$p_2 = p_{11} + p_{21}$		$\sum_{i,j} p_{ij} = 1$

Table 1: Overview of involved quantities in a matched pairs experiment.

confidence interval for the difference of two independent proportions, however, pointed out by a referee, the coverage probability of a 95% confidence interval, when the numbers of trials in two independent binomial experiments are 3 and 4, respectively, is equal to 0.8734 when the two true proportions are equal to 0.3 and 0.5, respectively.

In the rest of the article, we discuss how to compute intervals for the difference of two paired proportions θ_P defined in (1), then describe the results for the difference of two independent proportions θ_I given in (7).

Intervals for the difference of two paired proportions

Suppose there are n independent and identical trials in an experiment, and each trial is inspected by two criteria 1 and 2. By criterion i, each trial is classified as S_i (success) or F_i (failure) for $i = 1, 2$. The numbers of trials with outcomes (S_1, S_2), (S_1, F_2), (F_1, S_2) and (F_1, F_2) are the observations, and are denoted by N_{11}, N_{12}, N_{21} and N_{22}, respectively. Thus $\underline{X} = (N_{11}, N_{12}, N_{21})$ follows a multinomial distribution with probabilities p_{11}, p_{12}, p_{21}, respectively. Let $p_i = P(S_i)$ be the two paired proportions. The involved quantities are displayed in Table 1. The parameter of interest is the difference of p_1 and p_2:

$$\theta_P \overset{def}{=} p_1 - p_2 = p_{12} - p_{21}. \tag{1}$$

To make interval construction simpler, let $T = N_{11} + N_{22}$ and $p_T = p_{11} + p_{22}$. We consider intervals for θ_P of form $[L(N_{12}, T), U(N_{12}, T)]$, where (N_{12}, T) also follows a multinomial distribution with probabilities p_{12} and p_T. The simplified sample space is

$$S_P = \{(n_{12}, t) : 0 \le n_{12} + t \le n\}$$

with a reduced parameter space $H_P = \{(\theta_P, p_T) : p_T \in D(\theta_P), -1 \le \theta_P \le 1\}$, where $D(\theta_P) = \{p_T : 0 \le p_T \le 1 - |\theta_P|\}$. The probability mass function of (N_{12}, T) in terms of θ_P and p_T is

$$p_P(n_{12}, t; \theta_P, p_T) = \frac{n!}{n_{12}!t!n_{21}!} p_{12}^{n_{12}} p_T^t p_{21}^{n_{21}}.$$

Suppose a lower one-sided $1 - \alpha$ confidence interval $[L(N_{12}, T), 1]$ for θ_P is available. It can be shown that $[-1, U(N_{12}, T)]$ is an upper one-sided $1 - \alpha$ confidence interval for θ_P if

$$U(N_{12}, T) \overset{def}{=} -L(n - N_{12} - T, T), \tag{2}$$

and $[L(N_{12}, T), U(N_{12}, T)]$ is a two-sided $1 - 2\alpha$ interval for θ_P. Therefore, we focus on the construction of $L(N_{12}, T)$ only in this section. The R code will provide two (lower and upper) one-sided intervals and a two-sided interval, all are of level $1 - \alpha$. The first two are the smallest. The third is the intersection of the two smallest one-sided $1 - \alpha/2$ intervals. It may be conservative since the infimum coverage probability may be greater than $1 - \alpha$ due to discreteness.

An inductive order on S_P

Following Wang (2012), the construction of the smallest $1 - \alpha$ interval $[L(N_{12}, T), 1]$ requires a predetermined order on the sample space S_P. An order is equivalent to assigning a rank to each sample point, and this rank provides an order on the confidence limits $L(n_{12}, t)$'s. Here we define that a sample point with a small rank has a large value of $L(n_{12}, t)$, i.e., a large point has a small rank. Let $R(n_{12}, t)$ denote the rank of (n_{12}, t). Intuitively, there are three natural requirements for R:

1) $R(n, 0) = 1$,

2) $R(n_{12}, t) \le R(n_{12}, t - 1)$,

3) $R(n_{12}, t) \le R(n_{12} - 1, t + 1)$,

as shown in the diagram below:

$$\begin{tikzcd}
(n_{12}-1,t+1) \arrow[dr, phantom, "\rotatebox{-45}{$\leq$}\ 3)"] & \\
& (n_{12},t) \arrow[d, phantom, "\rotatebox{90}{$\leq$}\ 2)"] \\
& (n_{12},t-1)
\end{tikzcd}$$

Therefore, $R(n-1,1)=2$ and a numerical determination is needed for the rest of $R(n_{12},t)$'s. Wang (2012) proposed an inductive method to determine all $R(n_{12},t)$'s, which is outlined below.

Step 1: Point $(n,0)$ is the largest point. Let $R_1=\{(n,0)\}=\{(n_{12},t)\in S_P : R(n_{12},t)=1\}$.

...

Step k: For $k>1$, suppose the ranks, $1,\ldots,k$, have been assigned to a set of sample points, denoted by $S_k=\cup_{i=1}^{k}R_i$, where R_i contains the ith largest point(s) with a rank of i. Thus, S_k contains the largest through kth largest points in S_P. The order construction is complete if $S_{k_0}=S_P$ for some positive integer k_0, and R assumes values of $1,...,k_0$.

Step $k+1$: Now we determine R_{k+1} that contains the $(k+1)$th largest point(s) in S_P.

Part a) For each point (n_{12},t), let $N_{(n_{12},t)}$ be the neighbor set of (n_{12},t) that contains the two points next to but smaller than (n_{12},t), see the diagram above. Let N_k be the neighbor set of S_k that contains all sets $N_{(n_{12},t)}$ for (n_{12},t) in S_k.

Part b) To simplify the construction on R, consider a subset of N_k, called the candidate set

$$C_k=\{(n_{12},t)\in N_k : (n_{12},t+1)\notin N_k, (n_{12}+1,t-1)\notin N_k\}, \tag{3}$$

from which R_{k+1} is going to be selected.

Part c) For each point $(n_{12},t)\in C_k$, consider

$$f^*_{(n_{12},t)}(\theta_P)=1-\alpha, \tag{4}$$

where

$$f^*_{(n_{12},t)}(\theta_P)=\inf_{p_T\in D(\theta_P)}\sum_{(n'_{12},t')\in(S_k\cup(n_{12},t))^c} p_P(n'_{12},t';\theta_P,p_T).$$

Let

$$L^*_P(n_{12},t)=\begin{cases}-1, & \text{if no solution for (4);}\\ \text{the smallest solution of (4),} & \text{otherwise.}\end{cases} \tag{5}$$

Then define R_{k+1} to be a subset of C_k that contains point(s) with the largest value of L^*_P. We assign a rank of $k+1$ to point(s) in R_{k+1} and let S_{k+1} be the union of R_1 up to R_{k+1}.

Since S_P is a finite set and S_k is strictly increasing in k, eventually, $S_{k_0}=S_P$ for some positive integer k_0 ($\leq (n+1)(n+2)/2$) and the order construction is complete.

The computation of the rank function $R(N_{12},T)$ in R

There are three issues to compute the rank function $R(n_{12},t)$:

i) compute the infimum in $f^*_{(n_{12},t)}(\theta_P)$;

ii) determine the smallest solution of equation (4);

iii) repeat this process on all points in S_P.

These have to be done numerically.

Regarding i), use a two-step approach to search for the infimum when p_T belongs to interval $D(\theta_P)$, i.e., in the first step, partition $D(\theta_P)$ into, for example, 30 subintervals, find the grid, say a, where the minimum is achieved; then in the second step, partition a neighborhood of a into, for example, 20 subintervals and search the minimal grid again. In total we compute about 50 ($=30+20$) function values. On the other hand, if one uses the traditional one-step approach, one has to compute 600 ($=30\times 20$) function values to obtain a similar result.

Regarding ii), the smallest solution is found by the bisection method with different initial upper search points and a fixed initial lower search point -1. The initial upper search point is the lower confidence limit of the previous larger point in the inductive search algorithm.

Regarding iii), use `unique()` to eliminate the repeated points in N_k and use `which()` to search for R_{k+1} from C_k (smaller) rather than N_k.

The smallest one-sided interval under the inductive order

For any given order on a sample space the smallest one-sided $1-\alpha$ confidence interval for a parameter of interest can be constructed following the work by Buehler (1957); Chen (1993); Lloyd and Kabaila (2003) and Wang (2010). This interval construction is valid for any parametric model. In particular, for the rank function $R(n_{12}, t)$ just derived, the corresponding smallest one-sided $1-\alpha$ confidence interval, denoted by $L_P(n_{12}, t)$, has a form

$$L_P(n_{12}, t) = \begin{cases} -1, & \text{if no solution for (6);} \\ \text{the smallest solution of (6),} & \text{otherwise,} \end{cases}$$

where

$$f_{(n_{12},t)}(\theta_P) = 1-\alpha \tag{6}$$

and

$$f_{(n_{12},t)}(\theta_P) = 1 - \sup_{p_T \in D(\theta_P)} \sum_{\{(n'_{12},t') \in S_P : R(n'_{12},t') \le R(n_{12},t)\}} p_P(n'_{12}, t'; \theta_P, p_T),$$

that are similar to (4) and (5).

Two facts are worth mentioning. a) Among all one-sided $1-\alpha$ confidence intervals of form $[L(N_{12}, T), 1]$ that are nondecreasing regarding the order by the rank function R, $L \le L_P$. So $[L_P, 1]$ is the best. b) Among all one-sided $1-\alpha$ confidence intervals of form $[L(N_{12}, T), 1]$, $[L_P, 1]$ is admissible by the set inclusion criterion (Wang, 2006). So $[L_P, 1]$ cannot be uniformly improved. These properties make $[L_P, 1]$ attractive for practice. The computation of L_P is similar to that of the rank function R.

Intervals for the difference of two independent proportions

Suppose we observe two independent binomial random variables $X \sim Bin(n_1, p_1)$ and $Y \sim Bin(n_2, p_2)$ and the difference

$$\theta_I = p_1 - p_2 \tag{7}$$

is the parameter of interest. The sample space $S_I = \{(x, y) : 0 \le x \le n_1, 0 \le y \le n_2\}$ consists of $(n_1+1)(n_2+1)$ sample points, the parameter space in terms of (θ_I, p_2) is $H_I = \{(\theta_I, p_2) : p_2 \in D_I(\theta_I), -1 \le \theta_I \le 1\}$, where $D_I(\theta_I) = \{p_2 : -\min\{0, \theta_I\} \le p_2 \le 1 - \max\{0, \theta_I\}\}$. The joint probability mass function for (X, Y) is

$$p_I(x, y; \theta_I, p_2) = \frac{n_1!}{x!(n_1-x)!}(\theta_I + p_2)^x (1-\theta_I-p_2)^{n_1-x} \frac{n_2!}{y!(n_2-y)!} p_2^y (1-p_2)^{n_2-y}.$$

Exact $1-\alpha$ confidence intervals for θ_I of form $[L(X, Y), 1]$, $[-1, U(X, Y)]$, $[L(X, Y), U(X, Y)]$ are of interest. Similar to (2), $U(X, Y) = -L(n_1 - X, n_2 - Y)$. Therefore, we only need to derive the smallest lower one-sided $1-\alpha$ confidence interval for θ_I, denoted by $[L_I(X, Y), 1]$. Then $U_I(X, Y) = -L_I(n_1 - X, n_2 - Y)$ is the upper limit for the smallest upper one-sided $1-\alpha$ interval.

An inductive order and the corresponding smallest interval

Following Wang (2010), a rank function $R_I(X, Y)$ is to be introduced on S_I. This function provides an order of the smallest one-sided interval $L_I(x, y)$. In particular, a point (x, y) with a small $R_I(x, y)$ is considered a large point and has a large value of $L_I(x, y)$. Similar to the rank function R in the previous section, R_I should satisfy three rules:

a) $R_I(n_1, 0) = 1$,

b) $R_I(x, y) \le R_I(x, y+1)$,

c) $R_I(x, y) \le R_I(x-1, y)$,

as shown in the diagram below:

$$\begin{tikzcd} & (x,y+1) \arrow[d, phantom, "\overset{\wedge}{|}\ b)"] \\ (x,y-1) \arrow[r, phantom, "\overset{c)}{\leq}"] & (x,y) \end{tikzcd}$$

Repeating the process in the previous section, we can derive this new rank function R_I on S_I and the corresponding smallest one-sided $1-\alpha$ confidence interval $[L_I(X,Y),1]$ for θ_I by replacing (n_{12},t) by (x,y), $D(\theta_P)$ by $D_I(\theta_I)$ and $p_P(N_{12},T;\theta_P,p_T)$ by $p_I(x,y;\theta_I,p_2)$. The only thing different is that for the case of $n_1=n_2=n$, R_I generates ties. For example, $R_I(x,y)=R_I(n-y,n-x)$ for any (x,y). However, the procedure developed is still valid for this case. Technical details were given in the Sections 2 and 3 of Wang (2010).

Examples

Example 1: Exact intervals for the difference of two paired proportions θ_P

We illustrate the usage of the `PairedCI()` function to calculate the exact smallest lower one-sided confidence interval $[L_P,1]$ for θ_P in (1) with the data from Karacan et al. (1976). In this study, 32 marijuana users are compared with 32 matched controls with respect to their sleeping difficulties, with $n_{11}=16, n_{12}=9, n_{21}=3$, and $n_{22}=4$. The second argument in the function is $t=n_{11}+n_{22}=20$.

Function `PairedCI()` has the following arguments:

```
PairedCI(n12, t, n21, conf.level = 0.95, CItype = "Lower", precision = 0.00001,
         grid.one = 30, grid.two = 20)
```

The arguments `n12`, `t`, and `n21` are the observations from the experiment. The value of `conf.level` is the confidence coefficient of the interval, $1-\alpha$, which is equal to the infimum coverage probability here. One may change the value of `CItype` to obtain either an upper one-sided or a two-sided interval. The precision of the confidence interval with a default value 0.00001 is rounded to 5 decimals. The values of `grid.one` and `grid.two` are the number of grid points in the two-step approach to search the infimum. The higher the values of `grid.one` and `grid.two`, the more accurate is the solution but the longer is also the computing time. Based on our extensive numerical study, we find that `grid.one = 30` and `grid.two = 20` are sufficient enough for the problem.

In the data by Karacan et al. (1976), the researchers wish to see how much more help the marijuana use provides for sleeping by using a lower one-sided 95% confidence interval $[L_P(n_{12},t),1]$ for $\theta_P=p_1-p_2$ at $(n_{12},t)=(9,20)$, where p_1 is the proportion of marijuana users who have sleeping improved, and p_2 is the proportion in the controls. Given that the package **ExactCIdiff** is installed to the local computer, type the following:

```
> library(ExactCIdiff)
> lciall <- PairedCI(9, 20, 3, conf.level = 0.95) # store relevant quantities
> lciall                  # print lciall
$conf.level
[1] 0.95                  # confidence level
$CItype
[1] "Lower"               # lower one-sided interval
$estimate
[1] 0.1875                # the mle of p1 - p2
$ExactCI
[1] 0.00613 1.00000       # the lower one-sided 95% interval
> lci <- lciall$ExactCI   # extracting the lower one-sided 95% interval
> lci                     # print lci
[1] 0.00613 1.00000
```

The use of marijuana helps sleeping because the interval $[0.00613,1]$ for θ_P is positive.

The upper one-sided 95% interval and the two-sided 95% interval for θ_P are given below for illustration purpose.

```
> uci <- PairedCI(9, 20, 3, conf.level = 0.95, CItype = "Upper")$ExactCI
> uci                     # the upper one-sided 95% interval
[1] -1.00000  0.36234
> u975 <- PairedCI(9, 20, 3, conf.level = 0.975, CItype = "Upper")$ExactCI
> u975                    # the upper one-sided 97.5% interval
```

```
[1] -1.00000  0.39521
> l975 <- PairedCI(9, 20, 3, conf.level = 0.975, CItype = "Lower")$ExactCI
> l975                  # the lower one-sided 97.5% interval
[1] -0.03564  1.00000
> ci95 <- PairedCI(9, 20, 3, conf.level = 0.95)$ExactCI
> ci95
[1] -0.03564  0.39521    # the two-sided 95% interval
                         # it is equal to the intersection of two one-sided intervals
```

In summary, three 95% confidence intervals, $[0.00613, 1]$, $[-1, 0.36234]$ and $[-0.03564, 0.39521]$, are computed for θ_P. Wang (2012) also provided R code to compute these three intervals, but the calculation time is about 60 times longer.

Example 2: Exact intervals for the difference of two independent proportions θ_I

The second data set is from a two-arm randomized clinical trial for testing the effect of tobacco smoking on mice (Essenberg, 1952). In the treatment (smoking) group, the number of mice is $n_1 = 23$, and the number of mice which developed tumor is $x = 21$; in the control group, $n_2 = 32$ and $y = 19$. The function `BinomCI()` computes exact confidence intervals for θ_I in (7), the difference of proportions between two groups.

Function `BinomCI()` has the following arguments:

```
BinomCI(n1, n2, x, y, conf.level = 0.05, CItype = "Lower", precision = 0.00001,
        grid.one = 30, grid.two = 20)
```

The arguments `n1`, `n2`, `x` and `y` are the observations from the experiment. The rest of the arguments are the same as in function `PairedCI()`.

In this clinical trial, the maximum likelihood estimate for the difference between two tumor rates θ_I is calculated as

$$\hat{\theta}_I = \frac{x}{n_1} - \frac{y}{n_2} = 0.319293.$$

The lower confidence interval $[L(X,Y), 1]$ for θ_I is needed if one wants to see that the treatment (smoking) increases the risk of tumor. Compute the interval by typing:

```
> lciall <- BinomCI(23, 32, 21, 19, CItype = "Lower")
> lciall                 # print lciall
$conf.level
[1] 0.95                 # confidence level
$CItype
[1] "Lower"
$estimate
[1] 0.319293             # the mle of p1 - p2
$ExactCI
[1] 0.133 1.00000        # the lower one-sided 95% interval
> lci <- lciall$ExactCI  # extracting the lower one-sided 95% interval
> lci
[1] 0.133 1.00000
```

The lower one-sided 95% confidence interval for θ_I is $[0.133, 1]$. Therefore, the tumor rate in the smoking group is higher than that of the control group.

The following code is for the upper one-sided and two-sided 95% confidence intervals.

```
> uci <- BinomCI(23, 32, 21, 19, conf.level = 0.95, CItype = "Upper")$ExactCI
> uci                    # the upper one-sided 95% interval
[1] -1.00000  0.48595
> u975 <- BinomCI(23, 32, 21, 19, conf.level = 0.975, CItype = "Upper")$ExactCI
> u975                   # the upper one-sided 97.5% interval
[1] -1.00000  0.51259
> l975 <- BinomCI(23, 32, 21, 19, conf.level = 0.975, CItype = "Lower")$ExactCI
> l975                   # the lower one-sided 97.5% interval
[1] 0.09468 1.00000
> ci95 <- BinomCI(23, 32, 21, 19)$ExactCI
> ci95
[1] 0.09468  0.51259     # the two-sided 95% interval
                         # it is equal to the intersection of two one-sided intervals
```

They are equal to $[-1, 0.48595]$ and $[0.09468, 0.51259]$, respectively.

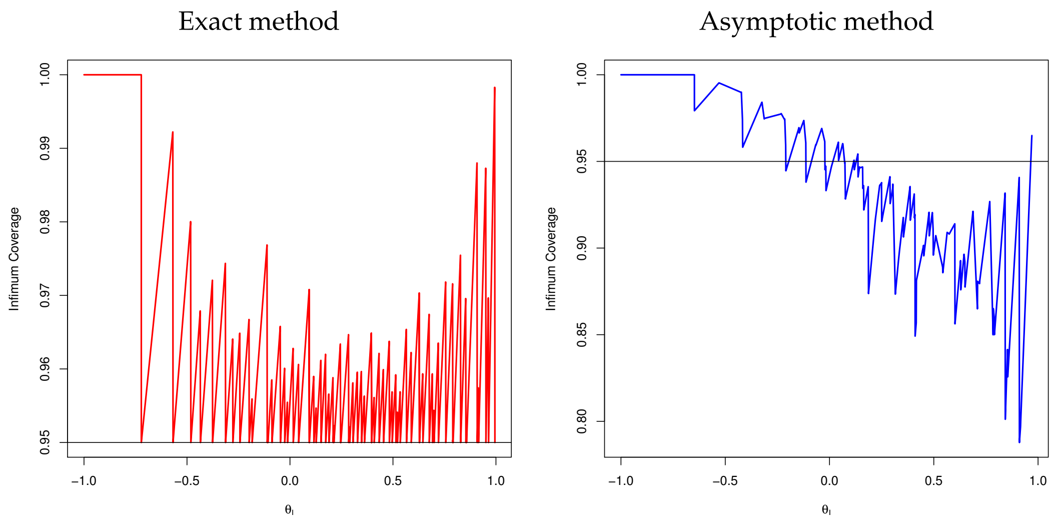


Figure 1: Coverage probability of upper confidence intervals for θ_I when $n_1 = n_2 = 10$ and $\alpha = 0.05$.

Comparison of results with existing methods

Our smallest exact one-sided confidence interval $[-1, U_I]$ for θ_I is first compared to an existing asymptotic interval (Newcombe, 1998b) using the coverage probability. The coverage of an upper confidence interval $[-1, U(X,Y)]$ as a function of θ_I is defined as:

$$Coverage(\theta_I) = \inf_{p_2 \in D_I(\theta_I)} P(\theta_I \le U(X,Y); \theta_I, p_2).$$

Ideally, a $1-\alpha$ interval requires that $Coverage(\theta_I)$ is always greater than or equal to $1-\alpha$ for all the possible values of θ_I.

The coverage for the exact upper 95% confidence interval $[-1, U_I]$ and the asymptotic upper confidence interval based on the tenth method of Newcombe (1998b), which is the winner of his eleven discussed intervals, is shown in Figure 1. The two intervals are calculated by `BinomCI()` and the function `ci.pd()` in the package **Epi**. The left plot of Figure 1 shows the coverage against $\theta_I \in [-1, 1]$ based on our exact method. As expected, it is always at least 95%. However, the coverage for the asymptotic interval may be much less than 95% as seen in the right plot of Figure 1. The coverage of the majority of θ_I values is below 95% and the infimum is as low as 78.8% for a nominal level of 95%. The similar results are observed for the asymptotic confidence intervals based on other methods, including the one proposed by Agresti and Caffo (2000).

In light of the unsatisfied coverage for the asymptotic approaches, we next compare our exact intervals to the exact intervals by the `PROC FREQ` procedure in the software SAS. First revisit Example 2, where SAS provides a wider exact two-sided 95% interval $[0.0503, 0.5530]$ for θ_I using the `EXACT RISKDIFF` statement within `PROC FREQ`. This is the SAS default. The other exact 95% interval in SAS using `METHOD = FMSCORE` is $[0.0627, 0.5292]$, which is narrower than the default but is wider than our two-sided interval. Also SAS does not compute exact intervals for θ_P at all.

Two exact upper intervals produced by `BinomCI()` in the R package **ExactCIdiff** and the `PROC FREQ` procedure in SAS are shown in Figure 2. The smaller upper confidence interval is preferred due to the higher precision. Almost all the points in the figure are below the diagonal line, which confirms a better performance of the interval by `BinomCI()`. The average lengths of the two-sided interval for $n_1 = n_2 = 10$ and $\alpha = 0.1$ are 0.636 and 0.712, respectively, for our method and the SAS default procedure. The newly developed exact confidence intervals have better performance than other asymptotic or exact intervals due to their guarantee on the coverage or because of their shorter length.

Summary

A group of three exact confidence intervals (lower one-sided, upper one-sided, and two-sided) are computed efficiently with the R package **ExactCIdiff** for each of the differences of two proportions: θ_P and θ_I. Each one-sided interval is admissible under the set inclusion criterion and is the smallest in a certain class of intervals that preserve the same order of the computed interval. Unlike asymptotic

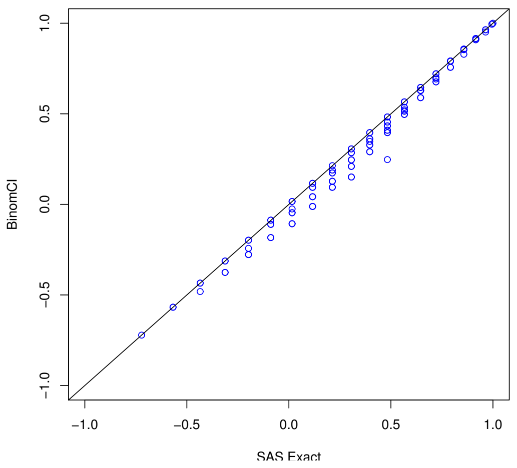


Figure 2: Exact upper confidence intervals for θ_I by `BinomCI()` and `PROC FREQ` when $n_1 = n_2 = 10$ and $\alpha = 0.05$.

intervals, these intervals assure that the coverage probability is always not smaller than the nominal level.

A practical issue for **ExactCIdiff** is the computation time that depends on the sample size $n = n_{12} + t + n_{21}$ for `PairedCI()` ($n = n_1 + n_2$ for `BinomCI()`) and the location of observations (n_{12}, t, n_{21}) ((x, y) for `BinomCI()`), e.g., `PairedCI(30,40,30) = [-0.15916,0.15916]`, with a sample size of 100, takes about a hour to complete on an HP laptop with Intel(R) Core(TM) i5=2520M CPU@2.50 GHz and 8 GB RAM, and `PairedCI(300,10,10,CItype = "Lower") = [0.86563,1.00000]`, with a sample size of 320, takes less than one minute. Our exact interval is constructed by an inductive method. By nature, when there are many sample points, i.e., the sample size is large, deriving an order on all sample points is very time consuming. Thus the confidence limit on a sample point, which is located at the beginning (ending) part of the order, needs a short (long) time to calculate. Roughly speaking, when the sample size is more than 100, one would expect a long computation time for a two-sided interval. More details may be found at: `http://www.wright.edu/~weizhen.wang/software/ExactTwoProp/examples.pdf.`

Acknowledgments

Wang's research is partially supported by NSF grant DMS-0906858. The authors are grateful to three anonymous referees and the Editor for their constructive suggestions.

Bibliography

A. Agresti and B. Caffo. Simple and effective confidence intervals for proportions and differences of proportions result from adding two successes and two failures. *The American Statistician*, 54(4): 280–288, 2000. [p69]

A. Agresti and B. A. Coull. Approximate is better than "exact" for interval estimation of binomial proportions. *The American Statistician*, 52(2):119–126, 1998. [p63]

L. D. Brown, T. T. Cai, and A. DasGupta. Interval estimation for a binomial proportion. 16(2):101–133, 2001. [p63]

R. J. Buehler. Confidence intervals for the product of two binomial parameters. *Journal of the American Statistical Association*, 52(280):482–493, 1957. [p66]

B. Carstensen, M. Plummer, E. Laara, and M. Hills. *Epi: A Package for Statistical Analysis in Epidemiology*, 2013. URL `http://CRAN.R-project.org/package=Epi`. R package version 1.1.44. [p63]

J. Chen. The order relations in the sample spaces and the confidence limits for parameters. *Advances in Mathematics*, 22(6):542–552, 1993. [p66]

J. M. Essenberg. Cigarette smoke and the incidence of primary neoplasm of the lung in the albino mouse. *Science*, 116(3021):561–562, 1952. [p68]

M. Fay. *exactci: Exact P-values and Matching Confidence Intervals for Simple Discrete Parametric Cases*, 2012. URL http://CRAN.R-project.org/package=exactci. R package version 1.2-0. [p63]

M. P. Fay. Two-sided exact tests and matching confidence intervals for discrete data. *R Journal*, 2(1): 53–58, 2010. URL http://journal.R-project.org/. [p63]

I. Karacan, A. Fernández-Salas, W. J. Coggins, W. E. Carter, R. L. Williams, J. I. Thornby, P. J. Salis, M. Okawa, and J. P. Villaume. Sleep electroencephalographic-electrooculographic characteristics of chronic marijuana users: Part I. *Annals of the New York Academy of Sciences*, 282:348–374, 1976. [p67]

C. Lloyd and P. Kabaila. Letter to the Editor: Some comments on "On construction of the smallest one-sided confidence interval for the difference of two proportions". *The Annals of Statistics*, 38(6): 3840–3841, 2010. [p63]

C. J. Lloyd and P. Kabaila. On the optimality and limitations of Buehler bounds. *Australian & New Zealand Journal of Statistics*, 45(2):167–174, 2003. [p66]

D. V. Mehrotra, I. S. Chan, and R. L. Berger. A cautionary note on exact unconditional inference for a difference between two independent binomial proportions. *Biometrics*, 59(2):441–450, 2003. [p63]

R. G. Newcombe. Improved confidence intervals for the difference between binomial proportions based on paired data. *Statistics in Medicine*, 17(22):2635–2650, 1998a. [p63]

R. G. Newcombe. Interval estimation for the difference between independent proportions: Comparison of eleven methods. *Statistics in Medicine*, 17(8):873–890, 1998b. [p63, 69]

SAS Institute Inc. *The SAS System, Version 9.3*. SAS Institute Inc., Cary, NC, 2011. URL http://www.sas.com/. [p63]

R. Scherer. *PropCIs: Various Confidence Interval Methods for Proportions*, 2013. URL http://CRAN.R-project.org/package=PropCIs. R package version 0.2-0. [p63]

G. Shan and W. Wang. *ExactCIdiff: Inductive Confidence Intervals for the difference between two proportions*, 2013. URL http://CRAN.R-project.org/package=ExactCIdiff. R package version 1.3. [p63]

D. Sun and H. J. Park. *ExactNumCI: Exact Confidence Interval for Binomial Proportions*, 2013. URL http://code.google.com/p/exactnumci/. R package version 1.0.0. [p63]

W. Wang. Smallest confidence intervals for one binomial proportion. *Journal of Statistical Planning and Inference*, 136(12):4293–4306, 2006. [p66]

W. Wang. On construction of the smallest one-sided confidence interval for the difference of two proportions. *The Annals of Statistics*, 38(2):1227–1243, 2010. [p63, 66, 67]

W. Wang. An inductive order construction for the difference of two dependent proportions. *Statistics & Probability Letters*, 82(8):1623–1628, 2012. [p63, 64, 65, 68]

W. Wang and Z. Zhang. Asymptotic infimum coverage probability for interval estimation of proportions. *Metrika*, in press. doi: 10.1007/s00184-013-0457-5. [p63]

Guogen Shan
Department of Environmental and Occupational Health
School of Community Health Sciences
University of Nevada Las Vegas,
Las Vegas, NV 89154, U. S. A.
guogen.shan@unlv.edu

Weizhen Wang (Corresponding author)
College of Applied Sciences
Beijing University of Technology
Beijing, 100124, P. R. China
and
Department of Mathematics and Statistics
Wright State University
Dayton, Ohio 45435, U. S. A.
weizhen.wang@wright.edu

rlme: An R Package for Rank-Based Estimation and Prediction in Random Effects Nested Models

by Yusuf K. Bilgic and Herbert Susmann

Abstract There is a lack of robust statistical analyses for random effects linear models. In practice, statistical analyses, including estimation, prediction and inference, are not reliable when data are unbalanced, of small size, contain outliers, or not normally distributed. It is fortunate that rank-based regression analysis is a robust nonparametric alternative to likelihood and least squares analysis. We propose an R package that calculates rank-based statistical analyses for two- and three-level random effects nested designs. In this package, a new algorithm which recursively obtains robust predictions for both scale and random effects is used, along with three rank-based fitting methods.

Introduction

Rank-based procedures retain distribution-free estimation and testing properties. These procedures are much less sensitive to outliers than the traditional analyses when random errors are not normally distributed. Alternative robust score functions can be accommodated with the rank-based methods to protect analyses from influential observations in factor and response spaces. Also, the choice of these score functions could depend on the prior knowledge on error distributions. The Wilcoxon score function is fairly efficient for moderate to heavy-tailed error distributions. For example, rank-based procedures with Wilcoxon scores achieve up to 95% efficiency relative to least squares methods when the data are normal and are much more efficient than least squares methods for heavy tailed error distributions. These properties make the rank-based methods appealing. However, to our knowledge, statistical analyses for random effects models using the rank-based methodology have not yet been considered in any statistical package. This article proposes an R package with three rank-based fitting methods that estimate fixed effects and predict random effects in two- and three-level random effects nested models.

The rank-based norm, analogous to the least squares norm, is briefly defined as

$$\|w\|_{\varphi} = \sum_{i}^{n} a\left[R(w_i)\right] w_i,\ w\epsilon R^n, \tag{1}$$

where the scores are generated as $a(i) = \varphi[i/(n+1)]$ for a non-decreasing function $\varphi(u)$, defined on the interval $(0,1)$, and $R(w_i)$ is the rank of w_i among $w_1, w_2, ..., w_n$. We assume without loss of generality that the scores sum to zero. Two of the most commonly used score functions are the Wilcoxon $\varphi(u) = \sqrt{12} \cdot (u - \frac{1}{2})$ and the sign $\varphi(u) = sgn[u\text{-}1/2]$.

The rank-based estimate of β for the independent error model $Y = X\beta + e$ is given by

$$\hat{\beta}_{\varphi} = \text{Argmin}\,\|Y - X\beta\|_{\varphi}\,. \tag{2}$$

Assume that the errors are independent and identically distributed with a continuous density function $f(x)$. Under regularity conditions, β is estimated by

$$\hat{\beta}\dot{\sim}N(\beta,\ \tau_{\varphi}^2(X^TX)^{-1}), \tag{3}$$

$$\tau_{\varphi} = \left[\int \varphi(u)\varphi_f(u)du\right]^{-1}, \tag{4}$$

with $\varphi_f(u) = -\frac{f'(F^{-1}(u))}{f(F^{-1}(u))}$. The parameter τ_{φ} is a scale parameter for the error terms e. The rank-based estimator of the fixed effects for independent linear models is asymptotically normal, shown in the work of Jaeckel and Jureckova in the 1970's. See Chapter 3, Hettmansperger and McKean (2011) for the relevant theory in detail. Recently, the **Rfit** package was released for rank-based regression analysis that uses rank-based norm and robust estimators for independent linear models (Kloke and McKean, 2012). We extend the rank-based regression methodology to include random effects nested models.

Random effects nested models are frequently utilized in many research areas such as: education, survey sampling, meta-analysis, agriculture, and health. Survey sampling might happen within organizational units, communities, clusters, or hospitals. The experimental design of interest is expressed in terms of fixed effects but, for these designs, nested factors are a natural part of the experiment. These nested effects are generally considered random and must be taken into account in the statistical analysis. For example, repeated measures design, randomized block design, and cluster correlated data consider each subject as a cluster having a correlated structure so the observations are nested within the subject. This subject would be a block, a cluster, an institution or a region. These designs are examples of two-level nested structures. Compound symmetric error structure is a natural variance-covariance form of these designs. It implies that all distinct members of a cluster or a subcluster are equally correlated with each other. When there is one more correlated level (subclusters) within clusters, this design could be called a three-level nested structure. Hierarchical linear models can also deal with these nested designs.

To illustrate the issues involved in data analysis, consider a simple example of a two-level nested design where students are nested within schools (see Model 5), and a three-level nested design where students are nested within sections in schools (see Model 6). Students are measured on a continuous univariate response variable of interest, y. The p-variate design data, including covariates, treatment, etc., are stored in the x vector. The problem is summarized in the linear model as

$$y_{ij} = \alpha + x_{ij}^T\beta + a_i + \epsilon_{ij}, \tag{5}$$

with $i = 1, ..., I$ and $j = 1, ..., n_i$, where a_i is the random effect for school i (cluster effects), I is the number of schools, n_i is the size of the i^{th} school, and

$$y_{ijk} = \alpha + x_{ijk}^T\beta + a_i + w_{j(i)} + \epsilon_{ijk}, \tag{6}$$

with $i = 1, ..., I$, $j = 1, ..., J_i$ and $k = 1, ..., n_{ij}$, where a_i is the random effect for school i (cluster effects), $w_{j(i)}$ is the random effect for the j^{th} section of school i (subcluster effects), J_i is the number of sections in school i, n_{ij} is the size of j^{th} section in school i. Random errors are uncorrelated and independent.

The main interest with these models would be to estimate the regression parameters β as fixed effects, to predict a_i and $w_{j(i)}$ as random effects, and scale parameters of the error and random effects. The *intra-class correlation coefficient* (ICC) for each nested level would be also estimated using the scale parameter estimates. ICC provides information on the degree of dependencies of the observations within the same cluster. It is a useful and contextual parameter associated with the random effects of clusters that measures the proportion of the variability of the response to the total variability. It is sometimes called *cluster effect* and applied only to random models. For example, independent observations within- and between-cluster yield an ICC of zero.

This nested analog could be adopted for other organizational studies and hierarchical data. These designs often address questions related to

- the examination of differences within and across clusters or contexts such as classrooms, schools, neighborhoods, or groups on individual outcomes;
- the investigation of the degree to which individuals within a group or cluster are similar as measured through the ICC;
- the study of the factors that explain institutional/cluster differences;
- the effects of clusters and treatment on individual scores, e.g., student's academic achievement -this design addresses both random and fixed effects-.

In this article, three rank-based fitting methods and a new prediction algorithm are briefly introduced. A data analysis example using our package, **rlme**, is also presented.

Three rank-based methods

This section introduces three rank-based fitting methods to obtain fixed effects estimations: Joint Ranking (JR), Generalized Rank Estimate (GR) and Generalized Estimating Equation (GEER). The algorithm for robust variance estimates and random effects predictions in random effects nested models, called Rank Prediction Procedure (RPP), is then introduced. To sketch the calculation algorithms in these methods, Models (5) and (6) can be rewritten in the general mixed model matrix and vector notations as follows:

$$Y = X\beta + e = X\beta + Zb + \epsilon, \tag{7}$$

where Y denotes a $n \times 1$ vector of responses, X is a $n \times (p + 1)$ known fixed effects design matrix, β is a $(p + 1) \times 1$ fixed effects parameter vector, Z is a $n \times k$ known random effects design matrix, b is a $k \times 1$ vector of random effects, and ϵ is a $n \times 1$ vector of random errors.

Alternatively, the model can be written in vectors of the observations obtained from independent I clusters. Within cluster k, let Y_k, X_k, and e_k denote respectively the $n_k \times 1$ vector of responses, the $n_k \times p$ design matrix, and the $n_k \times 1$ vector of errors. Then the general mixed model for Y_k is

$$Y_k = \alpha 1_{n_k} + X_k\beta + e_k,\ k = 1, ...I, \tag{8}$$

where the components of the random error vector e_k contain random effects and errors.

Joint ranking method: JR

This rank-based method for nested random effects models uses asymptotic results of the study by Kloke et al. (2009) in estimating fixed effects and standard errors. Kloke et al. (2009) developed the asymptotic theory for the rank-based estimates of fixed effects in the general mixed model using the general rank theory of Brunner and Denker (1994). The estimation of fixed effects in the JR method uses the dispersion function as in the independent linear model. However, the asymptotic distribution of $\hat{\beta}_{JR}$ has a different covariance matrix formula due to the correlated errors in the model. This is expressed as

$$\text{var}(\hat{\beta}_{JR}) \doteq \tau_\varphi^2 (X'X)^{-1}(\Sigma_\varphi)(X'X)^{-1}, \tag{9}$$

where $\Sigma_\varphi = \lim_{I\to\infty} \sum_{i=1}^{I} X_i'\Sigma_{\varphi,i}X_i \doteq \sum_{i=1}^{I} X_i'\Sigma_{\varphi,i}X_i$, $\Sigma_{\varphi,i}$ is given by $\Sigma_{\varphi,i} = \text{cov}(\varphi(F(e_i)))$, and $F(x)$ denotes the distribution function of errors.

After estimating the fixed effects in Model (7), we predict the nested random effects and estimate the variance components using the random prediction procedure explained in the next section. In this method, for each cluster in Model (8), simple moment estimators of $\Sigma_{\varphi,i}$ is calculated as in Kloke et al. (2009).

Iteratively reweighted generalized rank method: GR

The generalized rank-based fitting for the general mixed model is an iteratively reweighted rank method based on the Newton-type approximation. Hettmansperger and McKean (2011) developed the asymptotic properties of linearized rank estimators for use in the linear model with the k-step Gauss-Newton approximation without weights. Bilgic (2012) and Bilgic et al. (2013) extended this theory to the k-step GR method in the general mixed models. After the first fitting, the estimates are asymptotically equivalent to the independent case because residuals are no longer dependent because of the reweighting with covariance weights. This algorithm could work for any type of variance-covariance error structure in the general mixed models.

Consider Model (7) where Σ_Y is the variance-covariance matrix of the response vector Y and θ is the vector of variance components of the model. The proposed iteratively reweighted generalized rank-based algorithm is as follows:

(0) Set $l = 0$. The JR estimate serves as the initial fit.

(1) Obtain $\hat{\beta}^{(l)}$ as the rank-based fit of the model.

$$Y^* = X^*\beta + e^*, \tag{10}$$

where $Y^* = \hat{\Sigma}_Y^{-1/2}Y$, $X^* = \hat{\Sigma}_Y^{-1/2}X$, and $e^* = \hat{\Sigma}_Y^{-1/2}e$. Thus, $\hat{\beta}^{(l)}$ minimizes the rank norm with Y and X replaced by Y^* and X^*, respectively. If $l = 0$ then use $\hat{\Sigma}_Y = I_n$; otherwise use $\hat{\Sigma}_Y = \Sigma_Y(\hat{\theta}^{(l-1)})$.

(2) Use $\hat{\beta}^{(l)}$ to calculate the residuals, $\hat{e}^{(l)} = Y - X\hat{\beta}^{(l)}$.

(3) Use $\hat{e}^{(l)}$ to obtain $\hat{b}^{(l)}$, the predictor of b via the RPP algorithm.

(4) Use $\hat{b}^{(l)}$ to estimate the variance components, $\hat{\theta}^{(l)}$ via the RPP algorithm.

(5) If $\left\|\hat{\beta}^{(l)} - \hat{\beta}^{(l-1)}\right\| < TOL_1 \left\|\hat{\beta}^{(l-1)}\right\|$ and $\left\|\hat{\theta}^{(l)} - \hat{\theta}^{(l-1)}\right\| < TOL_2 \left\|\hat{\theta}^{(l-1)}\right\|$ then stop. Else let $\hat{\beta} = \hat{\beta}^{(l)}, \hat{\theta} = \hat{\theta}^{(l)}$ and $\hat{b} = \hat{b}^{(l)}$. Set $l = l + 1$ and return to step (1).

The estimators of the asymptotic variance-covariance matrix of $\hat{\beta}_{GR}$ require consistent τ_φ and Σ_Y which are obtained from the current estimate of weighted errors.

Rank-based generalized estimating equations method: GEER

Considering an alternative representation of generalized linear models for correlated data in estimates, Abebe et al. (2013) extended the general estimating equations (GEE) method of Liang and Zeger

(1986) for the general mixed models in the rank-based norm, and derived the asymptotic normality of the rank estimators. Briefly, as Abebe et al. (2013) describe, we can rewrite the general estimating equations expression proposed by Liang and Zeger in terms of the Euclidean norm in Model (11), and thus, the rank-based norm in Model (12) as follows:

$$D_{GEE}(\beta) = \sum_{i=1}^{I}(Y_i^* - D_i(\beta))^2 \tag{11}$$

and

$$D_{GEER}(\beta) = \sum_{i=1}^{I} \varphi \left[\frac{R(Y_i^* - D_i(\beta))}{n+1}\right] \cdot [Y_i^* - D_i(\beta)], \tag{12}$$

with $n_i \times 1$ vectors $Y_i^* = \hat{V}_i^{-1/2} \cdot Y_i$, the estimate of the covariance matrix of Y_i is $\hat{V}_i$, and $D_i(\beta) = \hat{V}_i^{-1/2} \cdot E[Y_i]$.

Abebe et al. (2013) developed a class of nonlinear robust estimators minimizing the rank-based norm of the residuals defined in the rank-based general estimating equations utilizing a 'working' covariance structure in the rank-based fitting as an analogue of the GEE. Thus, the estimate of β_{GEER} is obtained by the usual iterated reweighted least squares algorithm applied to the rank-based fitting in Equation (12).

Theory references and comparison

The asymptotic derivations for the proposed estimators are discussed in several papers; Kloke et al. (2009) for the JR method; Bilgic et al. (2013) for the GR method; and Abebe et al. (2013) for the GEER method. In these studies, the rank-based estimators are competitive with the traditional methods such as maximum likelihood (ML), restricted maximum likelihood (REML) and least squares in the normal case and outperform when random errors are contaminated and exhibit better efficiency properties of the estimates when outliers exist. Among the three methods, the JR method is unweighted so its empirical validity and efficiency for the fixed effects is reported to be poorer than the other two methods in the Monte Carlo study performed by Bilgic (2012). The empirical validity and efficiency of GR and GEER methods are reported to be very similar. The GR estimates and their standard errors are obtained from the rank-based norm properties, whereas the GEER combines the rank-based norm and least squares properties. For highly correlated data, the GR or GEER method would be preferred.

The **rlme** package uses the suite of R functions *ww* developed by Terpstra and McKean (2005) that computes fixed estimates for the rank analysis based on Wilcoxon scores when needed for independent linear models and initial fits. We plan to use the subroutines of the **Rfit** package for the next version of our package because it allows the user to choose the estimation algorithm for general scores.

Rank-based prediction procedure: RPP

So far, fixed effects estimations are calculated with the JR, GR and GEER methods described in the previous section. This section introduces robust predictions for random effects and variance components for two- and three-level nested designs. Robust predictions of random effects have been discussed in several papers, including Groggel (1983), Groggel et al. (1988), Dubnicka (2004), and Kloke et al. (2009). These predictions based on clusters use robust scale estimators. However, these papers only handle two-level random nested designs.

To illustrate how our recursive algorithm works, let us consider a two-level nested structure. The random effects model is defined as

$$y_{ij} = x_{ij}^T\beta + a_i + \epsilon_{ij} = x_{ij}^T\beta + e_{ij}, \tag{13}$$

for $i = 1,2,...,I$ and $j = 1,2,...,n_i$ (say, I schools, n_i students in each). a_i and ϵ_{ij} are random cluster effects and error effects, respectively. We observe the values of y_{ij} and x_{ij}, the variables a_i and ϵ_{ij} are not observable. In cluster i, we rewrite (13) as $y_{ij} - x_{ij}^T\beta = e_{ij} = a_i + \epsilon_{ij}$. This is a location model. The residuals $\hat{e}_{ij}$ obtained from one of rank-based fittings predict the cluster effects a_i. Let $\hat{a}$ be a consistent *location estimator*. The next step is that the residuals $\hat{\epsilon}_{ij}$ are obtained from the subtraction $\hat{\epsilon}_{ij} = \hat{e}_{ij} - \hat{a}_i$. Hence, we are ready to estimate the scale parameters of the errors with a robust *scale estimator*. In the package, RPP has two options for location and scale estimates. The options are the median (*med*) and Hodges-Lehman (*HL*) location estimators in Equation (14), and the median absolute deviation (*MAD*) and the dispersion scale estimator (*Disp*) in Equations (15) and (16), respectively. In the prediction algorithm, one estimator from each is needed. Natural pairs are *med-MAD* and *HL-Disp*.

For a vector e of errors e_i in $nx1$, the robust location estimate associated with the Wilcoxon scores is the HL estimator expressed as

$$HL(e) = \text{med}_{s \le t}\{(e_s + e_t)/2\}. \tag{14}$$

Our scale estimators in the package are MAD and $Disp$ defined as

$$MAD(e) = 1.483\text{med}_i|e_i - \text{med}_j\{e_j\}| \tag{15}$$

and

$$Disp(e) = \frac{2\sqrt{\pi}}{n}\sum_{i=1}^{n}\left(\frac{R(e_i)}{n+1} - \frac{1}{2}\right)\cdot e_i. \tag{16}$$

The estimator in Equation (16) is a consistent estimator of a scale parameter when the errors have a normal distribution (Hettmansperger and McKean, 2011). Kloke et al. (2009) suggest that the MAD is a robust and consistent estimator for scale parameter in clustered correlated designs.

For a three-level nested structure, consider Model (6). The residuals $\hat{e}_{ijk}$ obtained from the fit predict $a_i + w_{ij}$ using the location model

$$y_{ijk} - x_{ijk}^T\beta = e_{ijk} = a_i + w_{ij} + \epsilon_{ijk}. \tag{17}$$

To separate the cluster effects a_i and the subcluster effects w_{ij}, we need location estimates for each subcluster w_{ij}, which is nested within the cluster a_i. Proceeding over all subclusters using one of our robust location estimators, the estimates of w_{ij} are obtained. After subtracting these from the residuals $\hat{e}_{ijk}$, it yields the estimates of the cluster effects a_i using the location model $e_{ijk} - w_{ij} = a_i + \epsilon_{ijk}$. The scale parameters of the errors of each type, i.e. a_i, w_{ij} and ϵ_{ijk}, are then estimated with our robust scale estimators.

The RPP algorithm can handle k-level nestings in a hierarchical structure in the same manner. The algorithm needs only residuals from the model fitting for predictions of the random effects. In k-level random effects nested models, these residuals contain the estimates of errors and random effects. Groggel (1983) calls these random effects *pseudo-samples*, which are formed using consistent location estimators in this recursive way. Pseudo-samples are asymptotically equivalent to the true random effects in the model. See Bilgic (2012) for details.

Data example

The **rlme** package uses linear model syntax in the **lme4** package for two- and three-level models. To illustrate how our package does nested structured data analysis, a data set was obtained from the OECD Programme for International Student Assessment (PISA) conducted in 2009 (OECD, 2010). The data set includes 334 observations of metacognitive scores in 11 private schools in four geographic regions in USA. Metacognitive score is an index measure of the metacognitive aspect of learning. The research questions to be answered are how metacognitive scores depend on gender and age, and how the variability of the scores are explained by regional differences and school differences. Student scores are nested in the private schools which are nested within the regions. Data are correlated within region and school, hence, regions and schools are random effects on observations. This design would be considered hierarchical with two- or three- levels, such as students nested in regions, or students in schools nested within regions. In the package, this data set is called `schools`.

In our package, a 3-level nested design data analysis is done using the following syntax:

```
> library(rlme)
> data(schools)
> model = y ~ 1 + sex + age + (1 | region) + (1 | region:school)
> rlme.fit = rlme(model, schools, method="gr")

# For robust predictions, include rprpair="med-mad"
```

The formula syntax, the same as in the `lmer` function, expects the random effect terms to be enclosed with parenthesis, with nesting of variables denoted by a colon. In this example, region and school are two random effects, with school nested within region. The method is set to the rank-based method `gr` along with the prediction method `hl-disp` and Wilcoxon scores `wil` (the default). Alternatively the other rank-based fitting methods, `jr` or `geer`, and the maximum likelihood methods, `reml` or `ml`, may be called from `method`.

The calculated fit can be examined using the `summary` function:

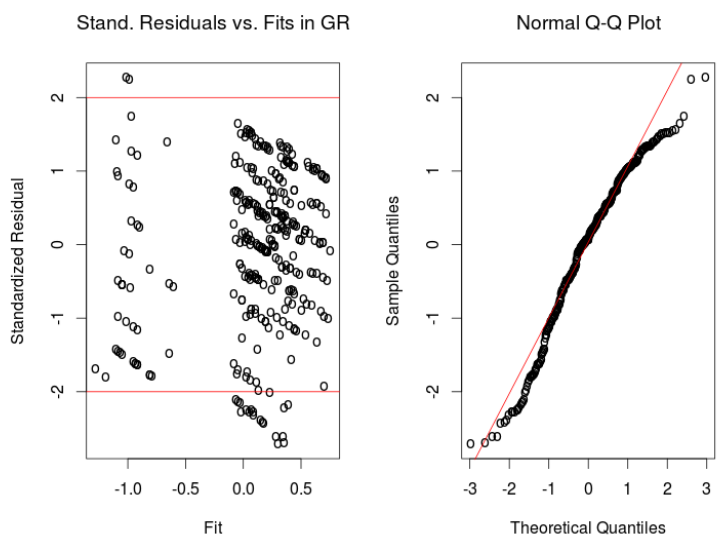


Figure 1: Standardized Residuals and QQ Plots.

```
> summary(rlme.fit)
Linear mixed model fit by  GR
Formula:  y ~ 1 + sex + age + (1 | region) + (1 | region:school)
Random effects:
 Groups         Name        Variance
 region:school  (Intercept) 0.14703510
 region         (Intercept) 0.01497416
 Residual                   0.81229310

Number of obs:
334 observations, 4 clusters, 11 subclusters

Fixed effects:
             Estimate   Std. Error  ..  p value
 (Intercept)  0.1586624 0.2686822   ..  0.554841676
 sex         -0.2953611 0.1071201   ..  0.005828259
 age          0.2260327 0.1621609   ..  0.163354085

Intra-class correlation coefficients
                 Estimates
 intra-cluster    0.1509132
 intra-subcluster 0.1662823

cov-var (fixed effects)
                           sex           age
     7.219013e-02 -0.0002000644 -1.672882e-05
sex -2.000644e-04  0.0114747092  9.802962e-04
age -1.672882e-05  0.0009802962  2.629616e-02
```

Here, *intra-cluster*, ρ_{region}, is the robust estimate of the intra-class correlation coefficient for region and *intra-subcluster*, $\rho_{school(region)}$, is for school nested within region. We can say that regional differences explain only 15.1% of the total variability in the model. The difference, $16.6\% - 15.1\% = 1.5\%$, shows that the contribution of the school variability to the total variability could be ignored. Using the `reml` method, this result is calculated around 2.1%, similar to the result of the rank analysis.

The plot function can be used to generate the standardized residuals vs. the fitted response and a normal Q-Q plot as shown in Figure 1:

```
> plot(rlme.fit)
```

The residuals and random effects can be extracted from the fit. For example, we can extract the raw residuals through the list element ehat:

```
# Raw residuals
> rlme.fit$ehat
[,1]
1   -1.186129739
2   -1.641494973
3   -1.147704177
...
334 -0.327186795
```

Several other elements of interest would include the effects estimates/predictions of the fixed effects, errors, clusters, and subclusters, which are obtained from `rlme.fit$fixed.effects`, `..$effect.err`, `..$effect.cluster`, and `..$effect.subcluster`, respectively. A full list can be found in the `help(rlme)` command or the `str` command. The same model can be also evaluated with the likelihood methods, `reml` or `ml`:

```
> rlme.fit = rlme(model, schools, method="reml")
> summary(rlme.fit)
Linear mixed model fit by  REML
...
Fixed effects:
             Estimate    Std. Error .. p value
 (Intercept) -0.08756901 0.2681410  .. 0.7439869
 sex         -0.26286182 0.1092493  .. 0.0161250
 age          0.24712016 0.1720753  .. 0.1509693

Intra-class correlation coefficients
                  Estimates
 intra-cluster    0.2433171
 intra-subcluster 0.2443941
...
```

The REML and GR results are slightly different, but do coincide in the inference at the 5% level. A 2-level random effects nested data analysis that students are nested within regions can be done in a similar syntax:

```
> data(schools)
> model = y ~ 1 + sex + age + (1 | region)
> rlme.fit = rlme(model, data = schools, method = "gr")
> summary(rlme.fit)
Linear mixed model fit by  GR
Formula:  y ~ 1 + sex + age + (1 | region)
...
Fixed effects:
             Estimate    Std. Error  .. p value
 (Intercept) -0.09298845 0.22061464  .. 6.733921e-01
 sex         -0.35939453 0.08977332  .. 6.245027e-05
 age          0.11882249 0.13665390  .. 3.845660e-01

Intra-class correlation coefficients
                  Estimates
 intra-cluster    0.1452482
 intra-subcluster 1.0000000
...
```

Diagnostics, `TDBETAS` and `CFITS`, to detect differences in fits for various methods can be obtained with the function `fitdvcov`. To compare the fixed effects estimates from any two fits, the covariance matrix from one rank-based method is required. Here we compare the REML and GR methods for the model:

```
> data(schools)
> model = y ~ 1 + sex + age + (1 | region) + (1 | region:school)
```

```
# Extract covariates into matrix
> cov = as.matrix(data.frame(rep(1, length(schools[,"y"])),
schools[,"sex"], schools[,"age"]))

# Fit the models using each method
> reml.fit = rlme(model, schools, method="reml")
> gr.fit = rlme(model, schools, method="gr")

# Extract fixed effects estimates
> reml.beta = reml.fit$fixed.effects$Estimate
> gr.beta = gr.fit$fixed.effects$Estimate

# Extract the covariance matrix of the fixed effects estimates
> var.b = reml.fit$var.b

> fitdvcov(cov, reml.beta, gr.beta, var.b)$tdbeta
[1] 0.9406339

# The following gets CFITS. For more info about diagnostics and
# benchmarks, see help(fitdvcov) and help(compare.fits)
> fitdvcov(cov, reml.beta, gr.beta, var.b)$cfits

# Graphing alone and getting standard residuals
> getgrstplot(gr.fit)
> getgrstplot(gr.fit)$sresid

> getlmestplot(reml.fit)
> getlmestplot(reml.fit)$sresid
```

In our package, in case of potential outliers in factor space, high breakdown weights (hbr), `weight="hbr"`, are specified along with `method="geer"`. We use the routines of the *ww* R codes for independent models:

```
> data(schools)
> rlme.fit = rlme(y ~ 1 + sex + age, data = schools)

You have entered an independent linear model.
Continuing using the ww package.

Wald Test of H0: BETA1=BETA2=0
TS: 38.4414 PVAL: 0

Drop Test of H0: BETA1=BETA2=0
TS: 7.819 PVAL: 5e-04

          EST     SE      TVAL PVAL
BETA0  0.2333 0.0000 9105.8041    0
BETA1 -0.4192 0.0478   -8.7664    0
BETA2  0.0000 0.0772    0.0000    1
```

Summary and further directions

The **rlme** package analyzes random effects nested models with respect to estimation, inference and prediction for two- and three-level designs. These designs are a class of mixed models. Hierarchical linear model users may be interested in our package for robust analysis as well.

In the **rlme** package, estimations and inferences are obtained from three different rank-based methods along with the likelihood methods obtained from the **nlme** package (Pinheiro et al., 2013). These three methods include the joint ranking (JR), iteratively reweighted generalized rank (GR) and rank-based generalized estimating equations (GEER). Variance estimates and random effects predictions are obtained along with a robust algorithm, called Rank Prediction Procedure (RPP). New rank-based estimators are employed in these algorithm and methods. Diagnostics are included in our package as well.

We are planning to extend statistical analysis of iteratively generalized rank-based methods using rank-norm properties to the general mixed models with various error structures. We would welcome any feedback and/or collaborations.

Acknowledgement

We would like to thank J. W. McKean for his invaluable guidance in theory and C. Leary for reviewing a preliminary version of this article.

Bibliography

A. Abebe, J. W. McKean, J. D. Kloke, and Y. K. Bilgic. Iterated reweighted rank-based estimates for GEE models. Under revision, 2013. [p74, 75]

Y. K. Bilgic. *Rank-based estimation and prediction for mixed effects models in nested designs*. PhD dissertation, Western Michigan University, Department of Statistics, 2012. [p74, 75, 76]

Y. K. Bilgic, J. W. McKean, J. D. Kloke, and A. Abebe. Iteratively reweighted generalized rank-based method in mixed models. Manuscript, 2013. [p74, 75]

E. Brunner and M. Denker. Rank statistics under dependent observations and applications to factorial designs. *Statist. Plan. Infer*, 42:353–378, 1994. [p74]

S. R. Dubnicka. A rank-based estimation procedure for linear models with clustered data. *Journal of Modern Applied Statistical Methods*, 3(1):39–48, 2004. URL http://www.amstat.org/publications/jcgs/. [p75]

D. J. Groggel. *Asymptotic nonparametric confidence intervals for the ratio of scale parameters in balanced one-way random effects models.* PhD dissertation, University of Florida, Department of Statistics, 1983. [p75, 76]

D. J. Groggel, D. Wackerly, and P. Rao. Nonparametric estimation in one-way random effects models. *Communications in Statistics: Simulation and Computation*, 17:887–903, 1988. [p75]

T. P. Hettmansperger and J. W. McKean. *Robust Nonparametric Statistical Methods*. Chapman & Hall, 2011. [p72, 74, 76]

J. Kloke and J. McKean. *Rfit: Rank Estimation for Linear Models*, 2012. URL http://CRAN.R-project.org/package=Rfit. R package version 0.14. [p72]

J. Kloke, J. W. McKean, and M. Rashid. Rank-based estimation and associated inferences for linear models with cluster correlated errors. *Journal of the American Statistical Association*, 104(485):384–390, 2009. [p74, 75, 76]

K. Y. Liang and S. L. Zeger. Longitudinal data analysis using generalized linear models. *Biometrika*, 73: 13–22, 1986. [p74]

J. Pinheiro, D. Bates, S. DebRoy, D. Sarkar, and R Core Team. *nlme: Linear and Nonlinear Mixed Effects Models*, 2013. R package version 3.1-109. [p79]

J. T. Terpstra and J. W. McKean. Rank-based analysis of linear models using R. *Journal of Statistical Software*, 14(7):1–26, 7 2005. ISSN 1548-7660. URL http://www.jstatsoft.org/v14/i07. [p75]

Yusuf K. Bilgic
SUNY Geneseo
1 College Circle, Geneseo, NY 14454, (585) 245-5384
USA
yusuf.k.bilgic@gmail.com

Herbert Susmann
SUNY Geneseo
1 College Circle, Geneseo, NY 14454, (585) 245-5384
USA
herbps10@gmail.com

Temporal Disaggregation of Time Series

by Christoph Sax and Peter Steiner

Abstract Temporal disaggregation methods are used to disaggregate low frequency time series to higher frequency series, where either the sum, the average, the first or the last value of the resulting high frequency series is consistent with the low frequency series. Temporal disaggregation can be performed with or without one or more high frequency indicator series. The package **tempdisagg** is a collection of several methods for temporal disaggregation.

Introduction

Not having a time series at the desired frequency is a common problem for researchers and analysts. For example, instead of quarterly sales, they only have annual sales. Instead of a daily stock market index, they only have a weekly index. While there is no way to fully make up for the missing data, there are useful workarounds: with the help of one or more high frequency indicator series, the low frequency series may be *disaggregated* into a high frequency series. For example, quarterly exports could help disaggregating annual sales, and a foreign stock market index could help disaggregating the stock market index at home.

Even when there is no high frequency indicator series, one still may want to disaggregate a low frequency series. While the accuracy of the resulting high frequency series will be low, it may still be worth doing so. For example, estimating a vector-autoregressive model requires all variables to have the same frequency. Having one bad high frequency series could still be preferable to the switch to a lower frequency.

The package **tempdisagg** (Sax and Steiner, 2013) implements the following standard methods for temporal disaggregation: Denton, Denton-Cholette, Chow-Lin, Fernandez and Litterman. On the one hand, **Denton** (Denton, 1971) and **Denton-Cholette** (e.g. Dagum and Cholette, 2006) are primarily concerned with movement preservation, generating a series that is similar to the indicator series whether or not the indicator is correlated with the low frequency series. Alternatively, these methods can disaggregate a series without an indicator. On the other hand, Chow-Lin, Fernandez and Litterman use one or several indicators and perform a regression on the low frequency series. **Chow-Lin** (Chow and Lin, 1971) is suited for stationary or cointegrated series, while **Fernandez** (Fernández, 1981) and **Litterman** (Litterman, 1983) deal with non-cointegrated series.

All disaggregation methods ensure that either the sum, the average, the first or the last value of the resulting high frequency series is consistent with the low frequency series. They can deal with situations where the high frequency is an integer multiple of the low frequency (e.g. years to quarters, weeks to days), but not with irregular frequencies (e.g. weeks to months).

Temporal disaggregation methods are widely used in official statistics. For example, in France, Italy and other European countries, quarterly figures of Gross Domestic Product (GDP) are computed using disaggregation methods. Outside of R, there are several software packages to perform temporal disaggregation: Ecotrim by Barcellan et al. (2003); a Matlab extension by Quilis (2012); and a RATS extension by Doan (2008). An overview of the capabilities of the different software programs is given in Table 1.[1]

The first section discusses the standard methods for temporal disaggregation and summarizes them in a unifying framework. Section 2 discusses the working and implementation of the **tempdisagg** package. Section 3 presents an illustrative example.

A framework for disaggregation

The aim of temporal disaggregation is to find an unknown high frequency series y, whose sums, averages, first or last values are consistent with a known low frequency series y_l (The subscript l denotes low frequency variables). In order to estimate y, one or more other high frequency indicator variables can be used. We collect these high frequency series in a matrix X. For the ease of exposition and without loss of generality, the terms annual and quarterly will be used instead of low frequency and high frequency hereafter.

The diversity of temporal disaggregation methods can be narrowed by putting the methods in a two-step framework: First, a preliminary quarterly series p has to be determined; second, the

[1]Currently, there is no support for temporal disaggreation methods with a contemporaneous constraint (Di Fonzo, 1994).

Methods	Ecotrim	Matlab add-on	RATS add-on	tempdisagg
Chow-Lin (max. log)	◑	●	●	●
Chow-Lin (min. RSS)	●	●	○	●
Fernández	●	●	●	●
Litterman (max. log)	◑	●	●	●
Litterman (min. RSS)	●	●	○	●
Denton-Cholette	◐	◑	○	●
Denton	○	◐	○	●
Contemporaneous constraint	◐	◐	○	○

Table 1: Software packages (●/◐/◑/○: full/partial/erroneous/no implementation).

differences between the annual values of the preliminary series and the annual values of the observed series have to be distributed among the preliminary quarterly series. The sum of the preliminary quarterly series and the distributed annual residuals yields the final estimation of the quarterly series, $\hat{y}$. Formally,

$$\hat{y} = p + Du_l\,. \tag{1}$$

D is a $n \times n_l$ distribution matrix, with n and n_l denoting the number of quarterly and annual observations, respectively. u_l is a vector of length n_l and contains the differences between the annualized values of p and the actual annual values, y_l:

$$u_l \equiv \; y_l - Cp\,. \tag{2}$$

Multiplying the $n_l \times n$ conversion matrix, C, with a quarterly series performs *annualization*. With two years and eight quarters, and annual values representing the sum of the quarterly values (e.g. GDP), the conversion matrix, C, is constructed the following way:[2]

$$C = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{3}$$

Equation (1) constitutes a unifying framework for all disaggregation methods. The methods differ in how they determine the preliminary series, p, and the distribution matrix, D. Table 2 summarizes the differences in the calculation of p and D. We will discuss them in turn.

Preliminary series

The methods of **Denton** and **Denton-Cholette** use a single indicator as their preliminary series:

$$p = X\,, \tag{4}$$

where X is a $n \times 1$ matrix. As a special case, a constant (e.g. a series consisting of only 1s in each quarter) can be embodied as an indicator, allowing for temporal disaggregation without high frequency indicator series.

The regression-based methods **Chow-Lin**, **Fernandez** and **Litterman** perform a Generalized Least Squares Regression (GLS) of the annual values, y_l, on the annualized quarterly indicator series, CX. In this case, X represents a $n \times m$ matrix, where m denotes the number of indicators (including a possible constant). For a given variance-covariance matrix, Σ, the GLS estimator, $\hat{\beta}$, is calculated in the standard way (the estimation of Σ is discussed below):

$$\hat{\beta}(\Sigma) = \left[X'C'(C\Sigma C')^{-1}CX\right]^{-1} X'C'(C\Sigma C')^{-1}y_l\,. \tag{5}$$

The critical assumption of the regression-based methods is that the linear relationship between the annual series CX and y_l also holds between the quarterly series X and y. Thus, the preliminary series is calculated as the fitted values of the GLS regression:

$$p = \hat{\beta}X\,. \tag{6}$$

[2]Generally, for annual values representing the sum of the quarterly values, C is constructed as $I_{n_l} \otimes [1,1,1,1]$, the Kronecker product of an identity matrix of size n_l and a transposed vector of length n/n_l (4, in the present case). If instead of sums, annual values are averages of the quarterly values, the transposed vector becomes $[0.25, 0.25, 0.25, 0.25]$. If annual values are equal to the first or the last quarterly value, the vector is $[1,0,0,0]$ or $[0,0,0,1]$, respectively.

Methods	p	D	Σ
`denton`	X	$\Sigma C'(C\Sigma C')^{-1}$	Σ_{D}
`denton-cholette`	X	D_{DC}	
`chow-lin-maxlog`, `chow-lin-minrss-ecotrim`, `chow-lin-minrss-quilis`	$\hat{\beta}X$	$\Sigma C'(C\Sigma C')^{-1}$	$\Sigma_{\text{CL}}(\rho)$
`litterman-maxlog`, `litterman-minrss`	$\hat{\beta}X$	$\Sigma C'(C\Sigma C')^{-1}$	$\Sigma_{\text{L}}(\rho)$
`fernandez`	$\hat{\beta}X$	$\Sigma C'(C\Sigma C')^{-1}$	$\Sigma_{\text{L}}(0)$

Table 2: Methods for temporal disaggregation.

Distribution matrix

With the exception of Denton-Cholette, the distribution matrix of all temporal disaggregation methods is a function of the variance-covariance matrix, Σ:

$$D = \Sigma C'(C\Sigma C')^{-1} . \tag{7}$$

The **Denton** methods minimize the squared absolute or relative deviations from a (differenced) indicator series, where the parameter h defines the degree of differencing. For the *additive* Denton methods and for $h = 0$, the sum of the squared absolute deviations between the indicator and the final series is minimized. For $h = 1$, the deviations of first differences are minimized, for $h = 2$, the deviations of the differences of the first differences, and so forth. For the *proportional* Denton methods, deviations are measured in relative terms.

For the additive Denton method with $h = 1$, the variance-covariance matrix has the following structure:

$$\Sigma_{\text{D}} = (\Delta'\Delta)^{-1} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 2 & \cdots & 2 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 2 & \cdots & n \end{bmatrix} , \tag{8}$$

where Δ is a $n \times n$ difference matrix with 1 on its main diagonal, -1 on its first subdiagonal and 0 elsewhere. For $h = 2$, $\Delta'\Delta$ is multiplied by Δ' from the left and Δ from the right side. For $h = 0$, it is the identity matrix of size n.

Denton-Cholette is a modification of the original approach and removes the spurious transient movement at the beginning of the resulting series. While generally preferable, the calculation of the distribution matrix, D_{DC}, does not fit into the simple framework (see Dagum and Cholette, 2006, pp. 136, for an extensive description).

Chow-Lin assumes that the quarterly residuals follow an autoregressive process of order 1 (AR1), i.e., $u_t = \rho u_{t-1} + \epsilon_t$, where ϵ is WN$(0, \sigma_\epsilon)$ (with WN denoting White Noise) and $|\rho| < 1$. The resulting covariance matrix has the following form:

$$\Sigma_{\text{CL}}(\rho) = \frac{\sigma_\epsilon^2}{1-\rho^2} \cdot \begin{bmatrix} 1 & \rho & \cdots & \rho^{n-1} \\ \rho & 1 & \cdots & \rho^{n-2} \\ \vdots & \vdots & \ddots & \vdots \\ \rho^{n-1} & \rho^{n-2} & \cdots & 1 \end{bmatrix} . \tag{9}$$

The estimation of Σ_{CL} thus requires the estimation of an AR1 parameter ρ, which will be discussed in the next section. The variance, σ_ϵ^2, cancels out and does not affect the calculation of neither D nor $\hat{\beta}$.

The remaining methods deal with cases when the quarterly indicators and the annual series are not cointegrated. **Fernandez** and **Litterman** assume that the quarterly residuals follow a non-stationary process, i.e. $u_t = u_{t-1} + v_t$, where v is an AR1 ($v_t = \rho v_{t-1} + \epsilon_t$, where ϵ is WN$(0, \sigma_\epsilon)$). Fernandez is a special case of Litterman, where $\rho = 0$, and, therefore, u follows a random walk. The variance-covariance matrix can be calculated as follows:

$$\Sigma_{\text{L}}(\rho) = \sigma_\epsilon^2 \left[\Delta' H(\rho)' H(\rho) \Delta\right]^{-1} , \tag{10}$$

where Δ is the same $n \times n$ difference matrix as in the Denton case; $H(\rho)$ is a $n \times n$ matrix with 1 on its main diagonal, $-\rho$ on its first subdiagonal and 0 elsewhere. For the special case of Fernandez, with

$\rho = 0$, the resulting covariance matrix has the following form:

$$\Sigma_{\mathrm{L}}(0) = \sigma_\epsilon^2 \cdot (\Delta'\Delta)^{-1} = \sigma_\epsilon^2 \cdot \Sigma_D \,. \tag{11}$$

Estimating the autoregressive parameter

There are several ways to estimate the autoregressive parameter ρ in the Chow-Lin and Litterman methods. An iterative procedure has been proposed by Chow and Lin (1971). It infers the parameter from the observed autocorrelation of the low frequency residuals, u_l.

In a different approach, Bournay and Laroque (1979, p. 23) suggest the maximization of the likelihood of the GLS-regression:

$$L(\rho, \sigma_\epsilon^2, \beta) = \frac{\exp\left[-\frac{1}{2} u_l' \, (C \Sigma C')^{-1} \, u_l\right]}{(2\pi)^{n_l/2} \cdot [\det (C \Sigma C')]^{1/2}} \,, \tag{12}$$

where u_l is given by Eq. (2) and (6). $\hat{\beta}$ turns out to be the GLS estimator from Eq. (5). The maximum likelihood estimator of the autoregressive parameter, $\hat{\rho}$, is a consistent estimator of the true value, thus it has been chosen as the default estimator. However, in some cases, $\hat{\rho}$ turns out to be negative even if the true ρ is positive. Thus, by default, **tempdisagg** constrains the optimization space for ρ to positive values.

A final approach is the minimization of the weighted residual sum of squares, as it has been suggested by Barbone et al. (1981):

$$RSS(\rho, \sigma_\epsilon^2, \beta) = u_l' \, (C \Sigma C')^{-1} \, u_l \,. \tag{13}$$

Contrary to the maximum likelihood approach, σ_ϵ^2 does not cancel out. The results are thus sensitive to the specification of Σ, with different implementations leading to different but inconsistent estimations of ρ.

The tempdisagg package

The selection of a temporal disaggregation model is similar to the selection of a linear regression model. Thus, `td`, the main function of the package, closely mirrors the working of the `lm` function (package **stats**), including taking advantage of the `formula` interface.[3]

```
td(formula, conversion = "sum", to = "quarterly", method = "chow-lin-maxlog",
   truncated.rho = 0, fixed.rho = 0.5, criterion = "proportional", h = 1,
   start = NULL, end = NULL, ...)
```

The left hand side of the formula denotes the low frequency series, the right hand side the indicators. If no indicator is specified, the right hand side must be set equal to `1`. The variables can be entered as time series objects of class `"ts"` or as standard vectors or matrices. If entered as `"ts"` objects, the resulting series will be `"ts"` objects as well.

The `conversion` argument indicates whether the low frequency values are sums, averages, first or last values of the high frequency values (`"sum"` (default), `"average"`, `"first"` or `"last"`, respectively). The `method` argument indicates the method of temporal disaggregation, as shown in Table 2 (see `?td` for a complete listing of methods). The `to` argument indicates the high frequency destination as a character string (`"quarterly"` (default) or `"monthly"`) or as a scalar (e.g. `2`, `7`, for year-semester or week-day conversion). It is only required if no indicator series is specified (Denton methods), or if standard vectors are used instead of time series objects. Finally, you can set an optional start or end date. This is identical to pre-processing the input series with `window`.

`td` returns an object of class `"td"`. The function `predict` computes the disaggregated high frequency series, $\hat{y}$. If the high frequency indicator series are longer than the low frequency series, the resulting series will be extrapolated.

The implementation of **tempdisagg** follows the same notation and modular structure as the exposure in the previous section. Internally, `td` uses the `optimize` function (package **stats**) to solve the one-dimensional optimization problem at the core of the Chow-Lin and Litterman methods. For GLS estimation, `td` uses an efficient and nummerically stable algorithm that is based on the *qr*-decomposition (Paige, 1979).

[3]There is no `data` argument in `td`, however. Because `td` is working with series of different length and frequencies, it is not possible to combine them in a single `"data.frame"`.

An example

Suppose we have an annual series and want to create quarterly values that sum up to the annual values. Panel 1 of Fig. 1 depicts annual sales of the pharmaceutical and chemical industry in Switzerland, `sales.a`, from which we want to create a quarterly series. The following example demonstrates the basic use of **tempdisagg**. It can also be run by `demo(tempdisagg)`.

The most simple method is `"denton-cholette"` without an indicator series. It performs a simple interpolation that meets the temporal additivity constraint. In R, this can be done the following way:

```
> library(tempdisagg)
> data(swisspharma)
> m1 <- td(sales.a ~ 1, to = "quarterly", method = "denton-cholette")
> predict(m1)
```

`td` produces an object of class `"td"`. The formula, `sales.a ~ 1`, indicates that our low frequency variable, `sales.a`, will be disaggregated with a constant, `1` (see `?formula` for the handling of the intercept in the formula interface). The resulting quarterly values of sales can be extracted with the `predict` function. As there is no additional information on quarterly movements, the resulting series is very smooth (Panel 2 of Fig. 1).

While this purely mathematical approach is easy to perform and does not need any other data series, the economic value of the resulting series may be limited. There might be a related quarterly series that follows a similar movement than sales. For example, we may use quarterly exports of pharmaceutical and chemical products, `exports.q` (Panel 3 of Fig. 1):

```
> m2 <- td(sales.a ~ 0 + exports.q, method = "denton-cholette")
```

Because we cannot use more than one indicator with the `"denton-cholette"` (or `"denton"`) method, the intercept must be specified as missing in the formula (`0`). Contrary to the first example, the `to` argument is redundant, because the destination frequency can be interfered from the time series properties of `exports.q`. Applying the `predict` function to the resulting model leads to a much more interesting series, as shown in Panel 4 of Fig. 1. As the indicator series is longer than the annual series, there is an extrapolation period, in which quarterly sales are forecasted.

With an indicator, the `"denton-cholette"` method simply transfers the movement of the indicator to the resulting series. Even if in fact there were no correlation between the two series, there would be a strong similarity between the indicator and the resulting series. In contrast, regression based methods transfer the movement only if the indicator series and the resulting series are actually correlated on the annual level. For example, a Chow-Lin regression of the same problem as above can be performed the following way:

```
> m3 <- td(sales.a ~ exports.q)
```

As `"chow-lin-maxlog"` is the default method, it does not need to be specified. Like with the corresponding `lm` method, `summary` produces an overview of the regression:

```
> summary(m3)

Call:
td(formula = sales.a ~ exports.q)

Residuals:
    Min      1Q  Median      3Q     Max 
-77.892  -7.711  -4.628   9.647  36.448 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept) 1.241e+01  1.493e+00   8.311 1.06e-09 ***
exports.q   1.339e-02  1.672e-04  80.111  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

'chow-lin-maxlog' disaggregation with 'sum' conversion
36 low-freq. obs. converted to 146 high-freq. obs.
Adjusted R-squared: 0.9946  AR1-Parameter:     0 (truncated)
```

There is indeed a strong correlation between exports and sales, as it has been assumed in the `"denton-cholette"` example above. The coefficient of `exports.q` is highly significant, and the very

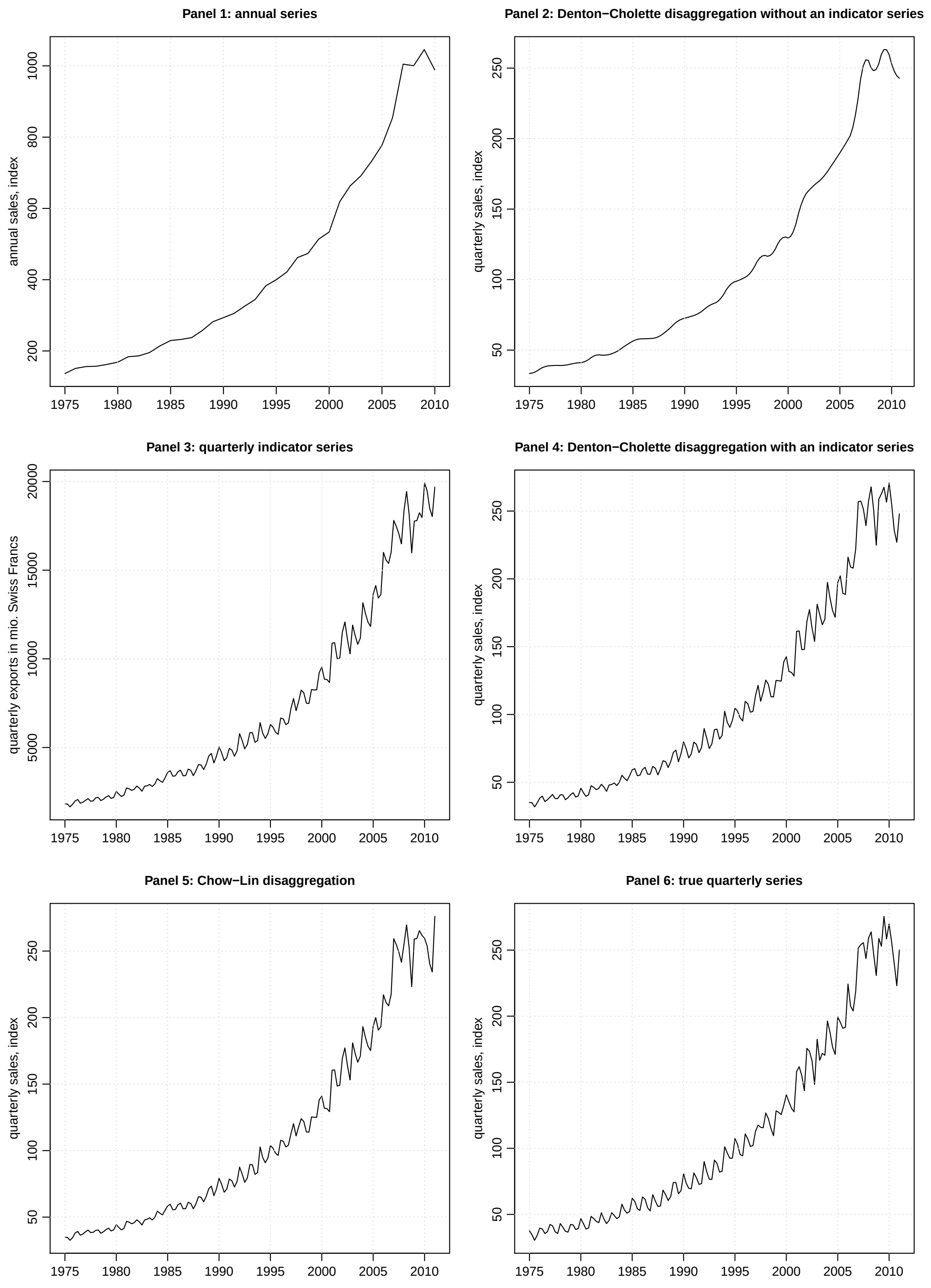


Figure 1: Disaggregating an annual series to quarterly series with no or one indicator series.

high adjusted R^2 points to a strong relationship between the two variables. The coefficients are the result of a GLS regression between the annual series. The AR1 parameter, ρ, was estimated to be negative; in order to avoid the undesirable side-effects of a negative ρ, it has been truncated to 0 (This feature can be turned off). Again, with the `predict` function, we can extract the resulting quarterly series of sales (Panel 5 of Fig. 1). Like all regression based methods, `"chow-lin-maxlog"` can also be used with more than one indicator series:

```
> m4 <- td(formula = sales.a ~ exports.q + imports.q)
```

In our example, we actually know the true data on quarterly sales, so we can compare the estimated values to the true values. With an indicator series, both the Denton method and Chow-Lin produce a series that is close to the true series (Panel 6 of Fig. 1). This is, of course, due to fact that in this example, exports are a good indicator for sales. If the indicator is less close to the series of interest, the resulting series will be less close to the true series.

Summary

tempdisagg implements the standard methods for temporal disaggregation. It offers a way to disaggregate a low frequency time series into a higher frequency series, while either the sum, the average, the first or the last value of the resulting high frequency series is consistent with the low frequency series. Temporal disaggregation can be performed with or without the help of one or more high frequency indicators. If good indicators are at hand, the resulting series may be close to the true series.

Bibliography

L. Barbone, G. Bodo, and I. Visco. Costi e profitti nell'industria in senso stretto: Un'analisi su serie trimestrali, 1970–1980. *Bolletino della Banca d'Italia*, pages 467–510, 1981. [p84]

R. Barcellan, T. Di Fonzo, D. Raffaele, V. Staplehurst, and D. Buono. *Ecotrim: A Program for Temporal Disaggregation of Time Series*, 2003. URL `https://circabc.europa.eu/w/browse/c6049bc0-c633-4cab-9811-b476ffe08370`. Version 1.01. [p81]

J. Bournay and G. Laroque. Réflexions sur la méthode d'élaboration des comptes trimestriels. *Annales de l'INSÉÉ*, 36:3–30, 1979. [p84]

G. C. Chow and A.-L. Lin. Best linear unbiased interpolation, distribution, and extrapolation of time series by related series. *The Review of Economics and Statistics*, 53(4):372–375, Nov. 1971. [p81, 84]

E. B. Dagum and P. A. Cholette. *Benchmarking, Temporal Distribution, and Reconciliation Methods for Time Series*. Lecture Notes in Statistics. Springer-Verlag, New York, 2006. [p81, 83]

F. T. Denton. Adjustment of monthly or quarterly series to annual totals: An approach based on quadratic minimization. *Journal of the American Statistical Association*, 66:99–102, Mar. 1971. [p81]

T. Di Fonzo. Temporal disaggregation of a system of time series when the aggregate is known: Optimal vs. adjustment methods. In *Workshop on Quarterly National Accounts*, pages 63–77, Paris, Dec. 1994. Eurostat. [p81]

T. Doan. *Disaggregate: A General Procedure for Interpolation*, 2008. URL `www.estima.com/procs_perl/disaggregate.src`. RATS library version Apr. 07, 2008. [p81]

R. B. Fernández. A methodological note on the estimation of time series. *The Review of Economics and Statistics*, 63(3):471–476, 1981. [p81]

R. B. Litterman. A random walk, Markov model for the distribution of time series. *Journal of Business & Economic Statistics*, 1(2):169–173, 1983. [p81]

C. C. Paige. Fast numerically stable computations for generalized linear least squares problems. *SIAM Journal on Numerical Analysis*, 16(1):165–171, 1979. [p84]

E. M. Quilis. *Temporal Disaggregation Library*, 2012. URL `www.mathworks.com/matlabcentral/fileexchange/24438-`. Matlab library version May 08, 2012. [p81]

C. Sax and P. Steiner. *tempdisagg: Methods for Temporal Disaggregation and Interpolation of Time Series*, 2013. URL `http://CRAN.R-project.org/package=tempdisagg`. R package version 0.22. [p81]

Christoph Sax
State Secretariat of Economic Affaires, Bern
University of Basel
Switzerland
christoph.sax@gmail.com

Peter Steiner
Federal Finance Administration, Bern
University of Bern
Switzerland
pete.steiner@gmx.net

Dynamic Parallelization of R Functions

by Stefan Böhringer

Abstract R offers several extension packages that allow it to perform parallel computations. These operate on fixed points in the program flow and make it difficult to deal with nested parallelism and to organize parallelism in complex computations in general. In this article we discuss, first, of how to detect parallelism in functions, and second, how to minimize user intervention in that process. We present a solution that requires minimal code changes and enables to flexibly and dynamically choose the degree of parallelization in the resulting computation. An implementation is provided by the R package **parallelize.dynamic** and practical issues are discussed with the help of examples.

The R language (Ihaka and Gentleman, 1996) can be used to program in the functional paradigm, *i.e.* return values of functions only depend on their arguments and values of variables bound at the moment of function definition. Assuming a functional R program, it follows that calls to a given set of functions are independent as long as their arguments do not involve return values of each other. This property of function calls can be exploited and several R packages allow to compute function calls in parallel, *e.g.* packages **parallel**, **Rsge** (Bode, 2012) or **foreach** (Michael et al., 2013; Revolution Analytics and Weston, 2013). A natural point in the program flow where to employ parallelization is where use of the `apply`-family of functions is made. These functions take a single function (here called the compute-function) as their first argument together with a set of values as their second argument (here called the compute-arguments) each member of which is passed to the compute-function. The calling mechanism guarantees that function calls cannot see each others return values and are thereby independent. This family includes the `apply`, `sapply`, `lapply`, and `tapply` functions called generically `Apply` in the following. Examples of packages helping to parallelize `Apply` functions include **parallel** and **Rsge** among others and we will focus on these functions in this article as well.

In these packages, a given `Apply` function is replaced by a similar function from the package that performs the same computation in a parallel way. Fixing a point of parallelism introduces some potential problems. For example, the bootstrap package **boot** (Davison and Hinkley, 1997; Canty and Ripley, 2013) allows implicit use of the **parallel** package. If bootstrap computations become nested within larger computations the parallelization option of the `boot` function potentially has to be changed to allow parallelization at a higher level once the computation scenario changes. In principle, the degree of parallelism could depend on parameter values changing between computations thereby making it difficult to choose an optimal code point at which to parallelize. Another shortcoming of existing solutions is that only a single `Apply` function gets parallelized thereby ignoring parallelism that spans different `Apply` calls in nested computations. The aim of this paper is to outline solutions that overcome these limitations. This implies that the parallelization process should be as transparent as possible, *i.e.* requiring as little user intervention as necessary. An ideal solution would therefore allow the user to ask for parallelization of a certain piece of code and we will try to approximate this situation. Potential benefits for the user are that less technical knowledge is required to make use of parallelization, computations can become more efficient by better control over the scaling of parallelization, and finally programs can better scale to different resources, say the local machine compared to a computer cluster.

This article is organized as follows. We first give further motivation by an example that highlights the problems this approach seeks to address. We then outline the technical strategy needed to determine the parallelism in a given function call. After that, trade-offs introduced by such a strategy are discussed. We conclude by benchmarking two examples and discussing important practical issues such as deviations of R programs from the functional programming style.

Dynamic parallelism in R functions

Let us start by looking at an example that tries to condense real-world problems in short, self-contained code which illustrates issues to be solved. Regression analyses are permformed on the `iris` data set as follows.

Example 1

```
Lapply <- lapply
Sapply <- sapply

library(sets)
data(iris)
```

```
d <- iris; response <- 'Species'; R <- .01; Nl <- 1e1; Nu <- 1e5

vars <- setdiff(names(d), response)
responseLevels <- levels(d[[response]])

minimax <- function(v, min = -Inf, max = Inf)
  ifelse(v < min, min, ifelse(v > max, max, v))
N <- function(p, r = R)
  (2 * qnorm(p, lower.tail = FALSE)/r)^2 * (1 - p)/p
analysis <- function(data, vars) {
  f1 <- as.formula(sprintf('%s ~ %s', response, paste(vars, collapse = ' + ')));
  f0 <- as.formula(sprintf('%s ~ 1', response));
  a <- anova(glm(f0, data = data), glm(f1, data = data), test = 'Chisq')
  p.value <- a[['Pr(>Chi)']][[2]]
}
permute <- function(data, vars, f, ..., M) {
  ps <- Sapply(0:M, function(i, data, vars, f, ...) {
    if (i > 0) data[, vars] <- data[sample(nrow(data)), vars];
    f(data, vars, ...)
  }, data, vars, f, ...)
  p.data <- ps[1]
  ps <- ps[-1]
  list(p.raw = p.data, p.emp = mean(ps[order(ps)] < p.data))
}
subsetRegression <- function() {
  r <- Lapply(responseLevels, function(l) {
    subsets <- as.list(set_symdiff(2^as.set(vars), 2^set()))
    r1 <- Sapply(subsets, function(subset) {
      d[[response]] = d[[response]] == l
      p.value <- analysis(d, unlist(subset))
      unlist(permute(d, unlist(subset), analysis,
        M = as.integer(minimax(N(p.value), Nl, Nu))))
    })
    output <- data.frame(subset = sapply(subsets, function(s)
      paste(s, collapse = '+')), t(r1))
  })
  names(r) <- responseLevels
  r
}
print(subsetRegression())
```

Variable Species is dichotomized for all of its levels and a subset analysis is performed by regressing these outcomes on all possible subsets of the other variables (function analysis). Also a permutation based P-value is computed (function permute) and the number of iterations depends on the raw P-value (p_{raw}) from the original logistic regression. Here, the number of iterations is chosen to control the length of the confidence interval for the permutation P-value (ci_l, ci_u) so that $(ci_u - ci_l)/p_{\text{raw}} < r$ (in probability), where r is a chosen constant (function N). To ensure robustness, the resulting number is constrained within an integer interval (function minimax).

Analyzing computational aspects of this code, we first note that our most global models are represented by response levels, in this case three, constituting a low level of parallelization. Second, the subset models vary in size, in this case by a factor of four. Third, the parallelism of permutation computations is data dependent and cannot be determined beforehand. It is thus not straightforward to choose a good point at which to parallelize. Finally, observe that we have copied the symbols sapply and lapply to upper-cased symbols and used them in places where parallelization is desirable. The sapply used for computing the variable output has not been marked in this way as it constitutes a trivial computation. The remainder of the article is concerned with achieving the goals stated above for a program for which desirable parallelization has been marked as in the code above.

Dynamic analysis

Parallelism in programs can be detected by static analysis of source code (*e.g.* Cooper and Torczon, 2011) or by dynamic analysis (*e.g.* Ernst, 2003) the latter relying on execution of the code at hand and the analysis of data gleaned from such executions. Example 1 motivates the use of dynamic analysis and we discuss static analysis later. In cases where parallelism is data dependent, dynamic analysis is

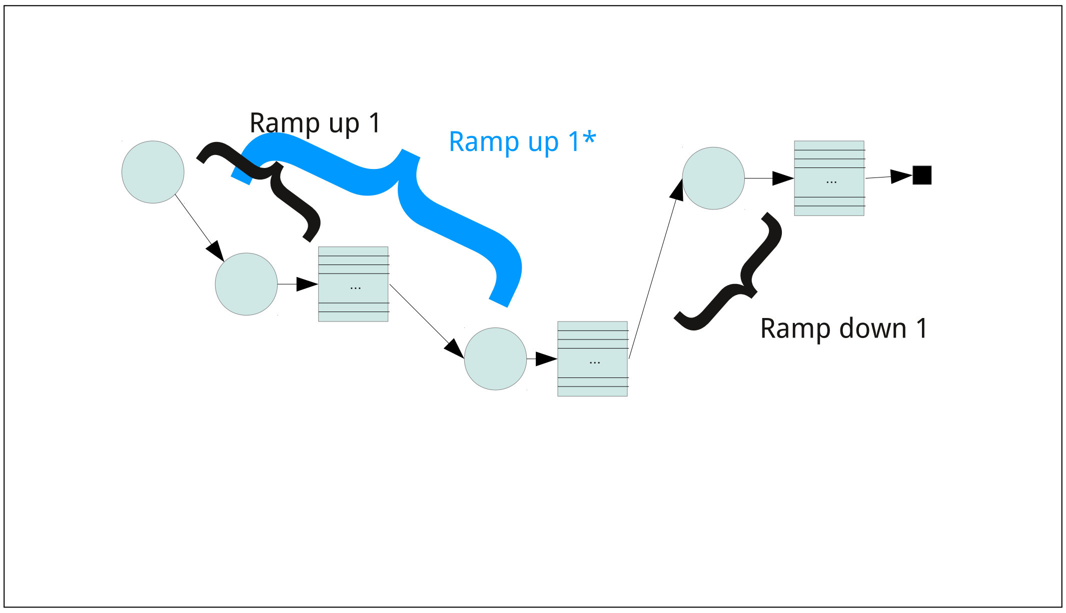


Figure 1: Abstraction of program flow with the following symbol semantics. Circles: entry points into or return points from functions; downward arrows: function calls; tables: `Apply` function calls; upward arrows: return path from functions; square: end of computation. Further explanation in text.

the only means to precisely determine the level of parallelism. On the other hand also in cases where parallelism is known or can be determined by inexpensive computations, dynamic analysis has the advantage of convenience as the user is not responsible for making decisions on parallelization.

In the following, dynamic analysis is performed on `Apply` functions marked as in Example 1. The overarching idea is to run the program but stop it in time to determine the degree of parallelism while still having spent only little computation time. Like in existing packages the assumption is made that the functional style is followed by the called functions, *i.e.* they do not exert side-effects nor depend on such.

Abstract program flow

Figure 1 depicts the program flow as seen in the parallelization process. Given code becomes a sequence of linear code (circles) leading to an `Apply` function, the `Apply`-function (table), and linear code executed thereafter (circles). This pattern repeats whenever `Apply` functions are nested. For now, we ignore the case where `Apply` functions are called sequentially as this case is not interesting for understanding the algorithm. We call code leading to a given `Apply` call the *ramp-up* and code executed after the `Apply` the *ramp-down* such that every program can be seen as an execution *ramp-up* – `Apply` – *ramp-down*. The task of dynamic analysis is to select the "best" `Apply` function, then separate execution into *ramp-up* – `Apply` – *ramp-down*, and perform the computation. Then, calls resulting from the selected `Apply` can be computed in parallel.

Algorithm

We now outline an abstract algorithm for implementing this program flow. Specific details about R-specific behavior are given in the implementation section. The problem is solved by re-executing the code several times – a choice that is justified below. The re-executions involve custom `Apply` functions which replace the original implementations with the ability to return execution to higher level `Apply` functions without executing the ramp-down, namely code following the `Apply` (escaping). The following re-executions take place:

- Probing (ramp-up): determine the level of parallelism, stop
- Freezing (ramp-up): save calls from `Apply`s for parallel execution, stop
- Recovery (ramp-down): replace calls that were parallelized with stored results, continue execution

Between the freezing and recovery steps, parallel computations are performed.

Probing

Probing potentially involves several re-executions as parallelism is determined for increasing nesting levels. For a given nesting level, probing simply stores the number of elements passed to the `Apply` calls at the specified nesting level and returns to the higher level. The sum of these elements is the level of parallelism achievable at that nesting level. If higher degree of parallelism is desired probing is repeated at a deeper nesting level.

Freezing

After a nesting level is chosen in the probing step, execution is stopped again in `Apply` calls at that nesting level. The calls that the `Apply` would generate are stored as unevaluated calls in a so-called freezer object.

Parallel execution

Parallel execution is controlled by a backend object. Similarly to the **foreach** package, several options are available to perform the actual computations (*e.g.* **snow** Tierney et al., 2013, or batch queuing systems).

Recovery

During recovery, execution is stopped at the same position as in the freezing step. The time results computed during parallel execution are retrieved and returned instead of evaluating function calls. Finally the whole computation returns with the final result.

Corner cases

If the requested level of parallelism exceeds the available parallelism, the computation will already finish in the probing step. This is because the probing level exceeds the nesting level at some point and execution will not be stopped. In this case the computation is performed linearly (actually sub-linearly because of the repeated re-executions).

When all results have been retrieved in the recovery step, the algorithm can switch back to probing and parallelize `Apply` code sequential to the first hierarchy of `Apply` calls. If no such `Apply` calls are present probing will compute the result linearly thus not incurring a performance penalty.

Implementation of R package parallelize.dynamic

The parallelization implementation is split into a so-called *front-end* part which implements the algorithm described above and a *backend* part which performs parallel execution. Currently, there are backends for local execution (*local* backend) which executes linearly, parallel execution on *snow* clusters (*snow* backend), and a backend for execution on *Sun Grid Engine* or *Open Grid Scheduler* batch queuing systems (*OGSremote* backend). This is a similar approach to the **foreach** package and potentially code can be shared between the packages. Back-ends are implemented as S4-classes and we refer to the package documentation for details on how to implement new backends.

API

Dynamic analysis of parallelism is performed by making use of replacements of `Apply` functions similar to existing packages. In this implementation, replacement functions have the same name as replaced functions with an upper case first letter, referred to as `Apply` functions. The new functions have exactly the same programming interface and the same semantics (*i.e.* they compute the same result) as the replaced functions, which is a difference to similar packages. One advantage is that programs can be very quickly adapted for parallel execution and the mechanism can be turned of by re-instating the original function definitions for `Apply` functions as done in Example 1.

Global state

In order to perform probing, freezing and recovery `Apply` functions maintain a global state containing the current position in the program flow. This is defined by the nesting level and an index number

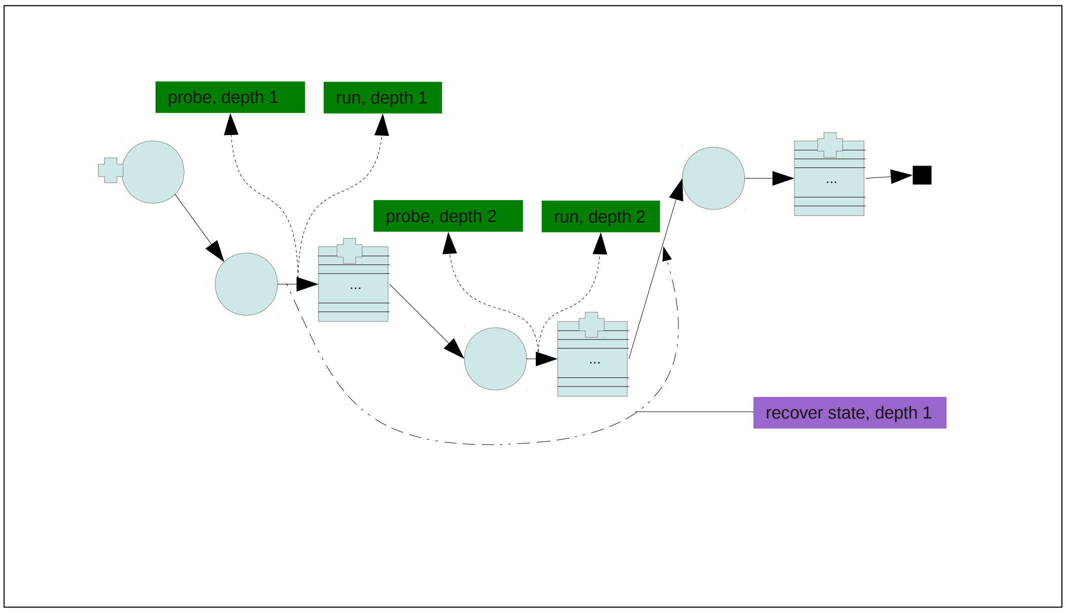


Figure 2: Exceptions used in the process of parallelization. Plus symbols indicate exception handlers and dotted lines indicate exceptions. See Figure 1 for explanation of other symbols.

counting `Apply` calls seen so far. Probing and freezing maintain an additional state. During probing, the number of elements passed to the `Apply` call under investigation is stored, during freezing, unevaluated calls resulting from the parallelized `Apply` call are stored.

Escaping

Probing and freezing need the ability to skip the ramp-down as they did not compute any results yet. This capability is implemented using the R exception handling mechanism defined by the functions `try`/ `stop`. Any `Apply` that needs to escape the ramp-down issues a call to `stop` that is caught by a `try` call that was issued in a higher-level `Apply`. As this disrupts normal program flow, execution can later not be resumed at the point where `stop` was called, requiring re-execution of the whole code. Figure 2 illustrates the use of exception handling.

Freezing

The freezing mechanism is defined by a reference class (`LapplyFreezer`) which stores unevaluated calls for parallel execution and results of these calls. The base class simply stores unevaluated calls and results as R objects. Subclasses that store results on disk (`LapplyPersistentFreezer`) or defer generation of individual calls from `Apply` calls to the parallel step (`LapplyGroupingFreezer`) exist and can be paired with appropriate backends.

Lexical scoping and lazy evaluation

R allows the use of variables in functions that are not part of the argument list (unbound variables). Their values are resolved by lexical scoping, *i.e.* a hierarchy of environments is searched for the first definition of the variable. This implies that all environments including the global environment would potentially have to be available when a parallel function call is executed to guarantee resolution of variable values. As for the *snow* and *OGS* backends execution takes place in different processes transferring these environments often constitutes unacceptable overhead. Therefore, if unbound variables are used with these backends the option `copy_environments` can be set to `TRUE` to force copying of environments. This mechanism constructs a new environment that only contains variables unbound in parallel function calls and computes their values using `get` in the correct environment. This is a recursive process that has to be repeated for functions called by compute-functions and is possibly expensive (compare Example 1). Potentially, these variables could be part of expressions that are evaluated lazily, *i.e.* values of these expressions should only be computed later when the expression is assigned to a variable. Code relying on the semantics of lazy evaluation could therefore work incorrectly.

A way to avoid copying of environments is to not use unbound variables in compute-functions. Functions called from compute-functions are allowed to contain unbound variables as long as they are bound by any calling function. The `copy_environments` option helps to minimize code changes to achieve parallelization but its use is not recommended in general (see discussion of Example 1 below).

Library and source dependencies

The `copy_environments` option can be used to ensure that function definitions are available in the parallel jobs. However, this mechanism avoids copying functions defined in libraries as libraries might contain initialization code containing side-effects upon which these functions could depend. Instead, required libraries have to be specified either as an element in the configuration list passed to `parallelize_initialize` or as the `libraries` argument of `parallelize_initialize`. Similarly, the `sourceFiles` component of the configuration list or the `sourceFiles` argument of `parallelize_initialize` specify R scripts to be sourced prior to computing the parallel job.

Examples

Example 1 continued

We continue Example 1 by extending it for use with package **parallelize.dynamic**. Parallelization is initialized by a call to `parallelize_initialize`. The following code has to replace the call to `subsetRegression` in Example 1.

```
library(parallelize.dynamic)
Parallelize_config <- list(
  libraries = 'sets',
  backends = list(snow = list(localNodes = 8, stateDir = tempdir()))
)
parallelize_initialize(Parallelize_config,
  backend = 'snow',
  parallel_count = 32,
  copy_environments = TRUE
)
print(parallelize_call(subsetRegression()))
```

It is good practice to put the definition of `Parallelize_config` into a separate file and describe all resources available to the user there. This file can then be sourced into new scripts and the call to `parallelize_initialize` can quickly switch between available resources by specifying the appropriate backend.

Backend	#Parallel jobs	Time	Speed-up
OGSremote	3	9129 sec	1.00
OGSremote	15	6073 sec	1.50
OGSremote	50	6206 sec	1.47

Table 1: *Time* is the absolute waiting time by the user averaged across two runs. *Speed-up* is relative to the first line.

The example is benchmarked on a four-core machine (*Intel Core i7*) running the *Open Grid Scheduler* (OGS; Open Grid Scheduler Development Team, 2013). In this example, we investigate the influence of varying the number of parallel jobs generated. Results are listed in Table 1 and times include all waiting times induced by polling job-statuses and wait times induced by OGS (on average each job waits 15 seconds before being started). The number of parallel jobs reflects parallelism in the program. For three jobs, each of the response levels is analyzed in parallel. Fifteen is the number of subsets of the covariates so that in this scenario the second level is parallelized (`Sapply` over the subsets). Fifty exceeds the number of subset-scenarios (45 subsets in total) so that the `Sapply` within `permute` is parallelized. Note, that in the last scenario execution is truly dynamic as the number of permutations depends on the generalized linear model (glm) computed on each subset. This implies that this glm is repeatedly computed during the re-executions, creating additional overhead. Choosing 15 jobs instead of three makes better use of the processing power so that speed-up is expected, a job count of 50 results in about the same wait time.

Increasing job counts beyond the number of parallel resources can increase speed if the parallel jobs differ in size and the overall computation depends on a single, long critical path. This path can potentially be shortened by splitting up the computation into more pieces. In this example, we could not benefit from such an effect. On the other hand, should we run the computation on a compute cluster with hundreds of available cores, we could easily create a similar amount of jobs to accommodate the new situation.

For the sake of demonstrating that the package can handle code with almost no modification, we allowed for unbound, global variables (*i.e.* functions use globally defined variables that are not passed as arguments). This forced us to use the `copy_environments = TRUE` option that makes the package look for such variables and include their definition into the job that is later executed. It is better practice in terms of using this package but also in terms of producing reproducible code in general to pass data and parameters needed for a computation explicitly as arguments to a function performing a specific analysis (analysis-function). We could also define all needed functions called from the analysis-function in separate files and list these under the `sourceFiles` key in the `Parallelize_config` variable. The package can then establish a valid compute environment by sourcing the specified files, loading listed libraries and transferring arguments of the analysis-function in which case we would not have to use the `copy_environments = TRUE` option. As a convenience measure the definition of the analysis-function itself is always copied by the package.

Example 2

The second example mimics the situation in Figure 1. It is more or less purely artificial and is meant to illustrate the overhead induced by the parallelization process.

```
parallel8 <- function(e) log(1:e) %*% log(1:e)
parallel2 <- function(e) rep(e, e) %*% 1:e * 1:e
parallel1 <- function(e) Lapply(rep(e, 15), parallel2)
parallel0 <- function() {
  r <- sapply(Lapply(1:50, parallel1), function(e) sum(as.vector(unlist(e))))
  r0 <- Lapply(1:49, parallel8)
  r
}

parallelize_initialize(Lapply_config, backend = 'local')
r <- parallelize(parallel0)
```

Backend	#Parallel jobs	Time
off	24	0.01 sec
local	24	0.14 sec
snow	24	19.80 sec
OGSremote	24	29.07 sec

Table 2: *Time* is the absolute waiting time by the user averaged across at least runs runs.

Again, a call to `parallelize_initialize` defines parameters of the parallelization and determines the backend. Function `parallelize` then executes the parallelization. Arguments to `parallelize` are the function to parallelize together with arguments to be passed to that function. Results from benchmark runs are shown in Table 2 and again absolute clock times are listed, *i.e.* time measured from starting the computation until the result was printed. *snow* and *OGSremote* backends were run on an eight core machine with Sun Grid Engine 6.2 installed with default settings. `parallelize` was configured to produce 24 parallel jobs. *off* in Table 2 denotes time for running without any parallelization. This can always be achieved by calling `parallelize_setEnable(F)` before `parallelize` which replaces the `Apply` functions by their native versions and `parallelize` by a function that directly calls its argument. The *local* backend performs parallelization but executes jobs linearly, thereby allowing to measure overhead. In this example overhead is ~ 0.13 seconds which is large in relative terms but unproblematic if the computation becomes larger. This overhead is roughly linear in the number of jobs generated. Comparing *snow* and *OGSremote* backends we can judge the setup time of these backends for their parallel jobs. It took *snow* a bit below a second and *OGSremote* a bit above a second to setup and run a job. It should be noted that the batch queuing characteristics are very influential for the *OGSremote* backend. This instance of the Sun Grid Engine was configured to run jobs immediately upon submission. This example does not transfer big data sets which would add to overhead, however, it seems plausible that an overhead of at most a couple of seconds per job makes parallelization worthwhile even for smaller computations in the range of many minutes to few hours.

Discussion & outlook

Limitations

Using the **parallelize.dynamic** package, existing code can be made to run in parallel with minimal effort. Certain workflows do not fit the computational model assumed here. Most notably the cost of the ramp-up determines the overhead generated by this package and might render a given computation unsuitable for this approach. In many cases re-factoring of the code should help to mitigate such overhead, however, this would render the point of convenience moot. It should also be pointed out that for a given computation for which time can be invested into choosing the code point at which to parallelize carefully and subsequently using packages like **parallel** or **foreach** should result in a more efficient solution.

Technical discussion

One way to reduce the cost of ramp-ups is to pull out code from nested loops and pre-compute their values, if possible. To help automate such a step, static code analysis can be used to separate computational steps from ramp-ups by analyzing code dependencies. Another option would be to extent the R language with an option to manipulate the "program counter", which would allow to resume code execution after a parallelization step in a very efficient manner. Such a change seems not straightforward but could also benefit debugging mechanisms.

All parallelization packages rely on function calls that are executed in parallel not to have side-effects themselves or to depend on such. It would be impractical to formally enforce this with the language features offered by R. Again, a language extension could enforce functional behavior efficiently, *i.e.* only the current environment (stack frame) may be manipulated by a function. For now, some care has to be taken by the user, however, this does not seem to be a big problem in practice. For most "statistical" applications such as simulations, bootstrapping, permutations or stochastic integration a reasonable implementation should naturally lead to side-effect free code.

Is a fully transparent solution possible? Replacing native `apply` functions directly with parallelization ones would have the benefit of requiring no modifications of the code at all. The current implementation would certainly suffer from too great an overhead, however, it is conceivable that profiling techniques (measuring computing time of individual function calls) could be used to gather prior knowledge on computational behavior of "typical" code allowing to exclude certain `apply` calls from the parallelization process. This seems a very challenging approach and requires extensive further efforts.

Conclusions

In the author's experience, most standard statistical workloads can easily be adapted to this parallelization approach and subsequently scale from the local machine to mid-size and big clusters without code modifications. Once standard configurations for the use of a local batch queuing system at a given site are created, this package can potentially dramatically broaden the audience that can make use of high performance computing.

As a final note, R is sometimes criticized for being an inefficient programming language which can be attributed to highly dynamic language features. The current implementation makes liberal use of many such features most notably introspection features to create unevaluated calls and perform their remote execution. The roughly 1000 lines of implementation (split roughly evenly between front-end and backend) demonstrate that these features are powerful and allow to execute a project such as this with a small code base. Also, the functional programming paradigm implemented in R allows for a natural attacking point of parallelization. This can be exploited to gain computational speed in a highly automatized way, a mechanism that is hard to imitate in procedural languages which traditionally have stakes in high performance computing.

Summary

In many practical situations it is straightforward to parallelize R code. This articles presents an implementation that reduces user intervention to a minimum and allows us to parallelize code by passing a given function call to the `parallelize_call` function. The major disadvantage of this implementation is the induced overhead which can often be reduced to a minimum. Advantages include that potentially little technical knowledge is required, computations can become more efficient by better control over the amount of parallelization, and finally that programs can be easily scaled to available resources. Future work is needed to reduce computational overhead and to complement this dynamic with a static analysis.

Bibliography

D. Bode. *Rsge: Interface to the SGE Queuing System*, 2012. URL `http://CRAN.R-project.org/package=Rsge`. R package version 0.6.3. [p89]

A. Canty and B. D. Ripley. *boot: Bootstrap R (S-Plus) Functions*, 2013. R package version 1.3-9. [p89]

K. Cooper and L. Torczon. *Engineering a Compiler*. Elsevier, Jan. 2011. ISBN 9780080916613. [p90]

A. C. Davison and D. V. Hinkley. *Bootstrap Methods and Their Applications*. Cambridge University Press, Cambridge, 1997. URL `http://statwww.epfl.ch/davison/BMA/`. ISBN 0-521-57391-2. [p89]

M. D. Ernst. Static and dynamic analysis: Synergy and duality. In *WODA 2003: ICSE Workshop on Dynamic Analysis*, pages 24–27, 2003. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.182.5350&rep=rep1&type=pdf#page=25`. [p90]

R. Ihaka and R. Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996. URL `http://www.tandfonline.com/doi/abs/10.1080/10618600.1996.10474713`. [p89]

K. Michael, J. W. Emerson, and S. Weston. Scalable strategies for computing with massive data. *Journal of Statistical Software*, 55(14):1–19, 2013. URL `http://www.jstatsoft.org/v55/i14`. [p89]

Open Grid Scheduler Development Team. Open Grid Scheduler: The official open source grid engine, 2013. URL `http://gridscheduler.sourceforge.net/`. [p94]

Revolution Analytics and S. Weston. *foreach: Foreach Looping Construct for R*, 2013. URL `http://CRAN.R-project.org/package=foreach`. R package version 1.4.1. [p89]

L. Tierney, A. J. Rossini, N. Li, and H. Sevcikova. *snow: Simple Network of Workstations*, 2013. URL `http://CRAN.R-project.org/package=snow`. R package version 0.3-13. [p92]

Stefan Böhringer
Leiden University Medical Center
Department of Medical Statistics and Bioinformatics
Postzone S-5-P, P.O.Box 9600
2300 RC Leiden
The Netherlands
`correspondence at s-boehringer.org`

CompLognormal: An R Package for Composite Lognormal Distributions

by S. Nadarajah and S. A. A. Bakar

Abstract In recent years, composite models based on the lognormal distribution have become popular in actuarial sciences and related areas. In this short note, we present a new R package for computing the probability density function, cumulative density function, and quantile function, and for generating random numbers of any composite model based on the lognormal distribution. The use of the package is illustrated using a real data set.

Introduction

Two-piece composite distributions arise in many areas of the sciences. The first two-piece composite distribution with each piece described by a normal distribution appears to have been used by Gibbons and Mylroie (1973). Recently, two-piece composite distributions with the first piece described a lognormal distribution (which we refer to as composite lognormal distributions) have proved popular in insurance and related areas. Cooray and Ananda (2005) introduced such distributions recently, but one of the referees has pointed out that composite lognormal distributions have been known before at least as early as Barford and Crovella (1998) in areas like modeling workloads of web servers.

Cooray and Ananda (2005) showed that composite lognormal distributions can give better fits than standard univariate distributions. They illustrate this fact for the Danish fire insurance data by introducing the *composite lognormal-Pareto distribution*, a distribution obtained by piecing together lognormal and Pareto probability density functions (pdfs). Scollnik (2007) improved the composite lognormal-Pareto distribution by using mixing weights as coefficients for each pdf replacing the constant weights applied earlier by Cooray and Ananda (2005). Scollnik (2007) also employed generalized Pareto distribution in place of Pareto distribution used by Cooray and Ananda (2005). Pigeon and Denuit (2011) provided further extensions of composite lognormal distributions by choosing the cutoff point (the point at which the two pdfs are pieced together) as a random variable. The most recent extensions of composite lognormal distributions are provided in Nadarajah and Bakar (2012).

Composite lognormal distributions have attracted considerable attention in spite of being introduced only in 2005. Some applications in the last three years include: estimation of insurance claim cost distributions (Bolancé et al., 2010), inflation and excess insurance (Fackler, 2010), large insurance claims in case reserves (Lindblad, 2011), statistical mechanics (Eliazar and Cohen, 2012), and modeling of mixed traffic conditions (Dubey et al., 2013).

The aim of this short note is to present a new contributed package **CompLognormal** for R that computes basic properties for *any* composite lognormal distribution. The properties considered include the pdf, cumulative distribution function (cdf), quantile function, and random numbers. Explicit expressions for these properties are given in Section "Properties". Two illustrations of the practical use of the new R package are given in Section "Illustrations".

Properties

Let $f_i(\cdot)$, $i = 1, 2$ be valid pdfs. Let $F_i(\cdot)$, $i = 1, 2$ denote the corresponding cdfs. In general, the pdf of a two-piece composite distribution is given by

$$f(x) = \begin{cases} a_1 f_1^*(x), & \text{if} \quad 0 < x \le \theta, \\ a_2 f_2^*(x), & \text{if} \quad \theta < x < \infty, \end{cases} \tag{1}$$

where $f_1^*(x) = f_1(x)/F_1(\theta)$, $f_2^*(x) = f_2(x)/\{1 - F_2(\theta)\}$, θ is the cutoff point, and a_1, a_2 are non-negative weights summing to one. As suggested in Nadarajah and Bakar (2012), we take $a_1 = \frac{1}{1+\phi}$ and $a_2 = \frac{\phi}{1+\phi}$ for $\phi > 0$.

The cdf corresponding to (1) is

$$F(x) = \begin{cases} \dfrac{1}{1+\phi} \dfrac{F_1(x)}{F_1(\theta)}, & \text{if} \quad 0 < x \le \theta, \\ \dfrac{1}{1+\phi} \left[1 + \phi \dfrac{F_2(x) - F_2(\theta)}{1 - F_2(\theta)} \right], & \text{if} \quad \theta < x < \infty. \end{cases} \tag{2}$$

The quantile function corresponding to (1) is

$$Q(u)=\begin{cases} F_1^{-1}\left(u(1+\phi)F_1(\theta)\right), & \text{if} \quad 0<u\le \dfrac{1}{1+\phi}, \\ F_2^{-1}\left(F_2(\theta)+(1-F_2(\theta))\left(\dfrac{u(1+\phi)-1}{\phi}\right)\right), & \text{if} \quad \dfrac{1}{1+\phi}<u<\infty. \end{cases} \tag{3}$$

This quantile function can be used to generate random numbers from the two-piece composite distribution.

We are interested in a two-piece composite distribution, where the first piece is specified by the lognormal distribution. So, we take

$$f_1(x)=\frac{1}{x\sigma}\psi\left(\frac{\ln x-\mu}{\sigma}\right) \qquad \text{and} \qquad F_1(x)=\Phi\left(\frac{\ln x-\mu}{\sigma}\right),$$

where $\psi(\cdot)$ and $\Phi(\cdot)$ denote the standard normal pdf and the standard normal cdf, respectively. For this choice, (1), (2), and (3) reduce to

$$f(x)=\begin{cases} \dfrac{\psi\left((\ln x-\mu)/\sigma\right)}{(1+\phi)\sigma x\Phi\left((\ln\theta-\mu)/\sigma\right)}, & \text{if} \quad 0<x\le\theta, \\ \dfrac{\phi f_2(x)}{(1+\phi)\left(1-F_2(\theta)\right)}, & \text{if} \quad \theta<x<\infty, \end{cases} \tag{4}$$

$$F(x)=\begin{cases} \dfrac{\Phi\left((\ln x-\mu)/\sigma\right)}{(1+\phi)\Phi\left((\ln\theta-\mu)/\sigma\right)}, & \text{if} \quad 0<x\le\theta, \\ \dfrac{1}{1+\phi}\left[1+\phi\dfrac{F_2(x)-F_2(\theta)}{1-F_2(\theta)}\right], & \text{if} \quad \theta<x<\infty, \end{cases} \tag{5}$$

and

$$Q(u)=\begin{cases} \exp\left\{\mu+\sigma\Phi^{-1}\left[u(1+\phi)\Phi\left(\dfrac{\ln\theta-\mu}{\sigma}\right)\right]\right\}, & \text{if} \quad 0<u\le\dfrac{1}{1+\phi}, \\ F_2^{-1}\left(F_2(\theta)+(1-F_2(\theta))\left(\dfrac{u(1+\phi)-1}{\phi}\right)\right), & \text{if} \quad \dfrac{1}{1+\phi}<u<\infty, \end{cases} \tag{6}$$

respectively. We shall refer to the distribution given by (4), (5), and (6) as the *composite lognormal distribution*. Random numbers from the composite lognormal distribution can be generated as

$$x_i=Q\left(u_i\right) \tag{7}$$

for $i=1,2,\ldots,n$, where u_i, $i=1,2,\ldots,n$ are random numbers from a uniform $[0,1]$ distribution. Further statistical properties of the composite lognormal distribution including moment properties can be found in Bakar (2012).

The pdf of two-piece composite distributions are in general not continuous or differentiable at the cutoff point θ. To have these properties satisfied, we impose the conditions $a_1 f_1^*(\theta)=a_2 f_2^*(\theta)$ and $a_1 df_1^*(\theta)/d\theta=a_2 df_2^*(\theta)/d\theta$. Nadarajah and Bakar (2012) have shown that these conditions are equivalent to the following for any composite lognormal distribution:

$$\mu=\ln\theta+\sigma^2+\theta\sigma^2\frac{f_2^{'}(\theta)}{f_2(\theta)}, \qquad \text{and} \qquad \phi=\frac{f_1(\theta)\left[1-F_2(\theta)\right]}{f_2(\theta)F_1(\theta)}. \tag{8}$$

The smoothness conditions in (8) are imposed for technical reasons—the respective parametric models then become more tractable.

Maximum likelihood estimation is a common method for estimation. Suppose we have a random sample $x_1,x_2,\ldots,x_n$ from (1). Let $\boldsymbol{\lambda}=(\lambda_1,\lambda_2,\ldots,\lambda_q)$ be the parameters of $f_2(\cdot)$. Suppose also that μ and ϕ can be expressed as $\mu=\mu(\sigma,\theta,\boldsymbol{\lambda})$ and $\phi=\phi(\sigma,\theta,\boldsymbol{\lambda})$, respectively. Then the log-likelihood function is

$$\begin{aligned} \ln L\left(\sigma,\theta,\boldsymbol{\lambda}\right) = & -n\ln(1+\phi)+\sum_{x_i\le\theta}\ln\psi\left(\frac{\ln x_i-\mu}{\sigma}\right)-\sum_{x_i\le\theta}\ln\left(\sigma x_i\right) \\ & -M\ln\Phi\left(\frac{\ln\theta-\mu}{\sigma}\right)+\sum_{x_i>\theta}\ln f_2\left(x_i\right) \\ & -m\ln\left[1-F_2(\theta)\right]+m\ln\phi, \end{aligned} \tag{9}$$

where $M=\sum_{i=1}^{n} I\{x_i\le\theta\}$ and $m=\sum_{i=1}^{n} I\{x_i>\theta\}$. It is clear that the maximum likelihood estimators of $(\sigma,\theta,\boldsymbol{\lambda})$ cannot be obtained in closed form. They have to be obtained numerically.

The new package **CompLognormal**, available from CRAN, computes (4), (5), and (6) for any given

Quantity	Calling sequence
$f(x)$ in (4)	`dcomplnorm(x,spec,sigma=1,theta=1,...)`
$F(x)$ in (5)	`pcomplnorm(x,spec,sigma=1,theta=1,...)`
$Q(u)$ in (6)	`qcomplnorm(p,spec,sigma=1,theta=1,...)`
x_i in (7)	`rcomplnorm(n,spec,sigma=1,theta=1,...)`

Table 1: Quantity and calling sequence for the composite lognormal distribution.

composite lognormal distribution. It also generates random numbers from the specified composite lognormal distribution. Table 1 summarizes the functions implemented in the package **CompLognormal** along with their arguments.

The input argument spec (a character string) specifies the distribution of the second piece of the composite lognormal distribution. The distribution should be one that is recognized by R. It could be one of the distributions implemented in the R base package or one of the distributions implemented in an R contributed package or one freshly written by a user. In any case, there should be functions dspec, pspec, qspec and rspec, computing the pdf, cdf, qf and random numbers of the distribution.

Some examples of spec are: spec = "norm", meaning that $f_2(x) = (1/\sigma)\psi((x-\mu)/\sigma)$ and $F_2(x) = \Phi((x-\mu)/\sigma)$; spec = "lnorm", meaning that $f_2(x) = \{1/(\sigma x)\}\,\psi((\ln x-\mu)/\sigma)$ and $F_2(x) = \Phi((\ln x-\mu)/\sigma)$; spec = "exp", meaning that $f_2(x) = \lambda\exp(-\lambda x)$ and $F_2(x) = 1-\exp(-\lambda x)$.

dcomplnorm, pcomplnorm, qcomplnorm and rcomplnorm can also take additional arguments in the form of .... These arguments could give inputs (for example, parameter values) for the distribution of the second piece of the composite lognormal distribution. For example, if spec = "norm" then ... can include mean = 1,sd = 1 to mean that $f_2(x) = \psi(x-1)$ and $F_2(x) = \Phi(x-1)$; if spec = "lnorm" then ... can include meanlog = 1,sdlog = 1 to mean that $f_2(x) = \psi(\ln x-1)$ and $F_2(x) = \Phi(\ln x-1)$; if spec = "exp" then ... can include rate = 1 to mean that $f_2(x) = \lambda\exp(-x)$ and $F_2(x) = 1-\exp(-x)$. Further details about the calling sequences can be seen from the documentation for the **CompLognormal** package.

The code for dcomplnorm, pcomplnorm and qcomplnorm currently uses constructs of the form

```
do.call(paste("d", spec, sep = ""), list(z}, \code{...))
```

This technique was necessary as in the R base package distributions are no data type but rather are given by the respective four constituent functions <prefix><name>, where <prefix> is one of r, d, p, q and <name> is the name of the distribution. A way to circumvent such constructions would be a data type "distribution". This has been available on CRAN for quite a while within the distr family of packages, with corresponding S4 classes for distributions. Recently, another approach based on S5-classes has been pursued in the package **poweRlaw** (Gillespie, 2013). We leave these as future work.

Illustrations

We describe two simple illustrations of the new package **CompLognormal**. The following packages should be loaded (after installing them if necessary) in advance:

```
library(CompLognormal)
library(actuar)
library(SMPracticals)
library(evd)
library(fitdistrplus)
library(stats4)
```

Illustration 1

The illustration presented here plots the pdf and the cdf of the composite lognormal-loglogistic distribution for varying parameter values.

```
curve(dcomplnorm(x, "llogis", 0.4, 0.5, shape = 1, scale = 0.8), xlim = c(0, 5),
      ylim = c(0, 0.7), xlab = "x", ylab = "f(x)", n = 250, col = "black", lty = 1)
d1 <- dcomplnorm(0.5, "llogis", 0.4, 0.5, shape = 1, scale = 0.8)
segments(0.5, 0, 0.5, d1, col = "black", lty = 2)
```

```
curve(dcomplnorm(x, "llogis", 0.3, 0.2, shape = 0.2, scale = 0.5), add = TRUE,
      col = "red", lty = 1)
d2 <- dcomplnorm(0.2, "llogis", 0.3, 0.2, shape = 0.2, scale = 0.5)
segments(0.2, 0, 0.2, d2, col = "red", lty = 2)
curve(dcomplnorm(x, "llogis", 0.5, 0.4, shape = 0.5, scale = 0.5), add = TRUE,
      col = "blue", lty = 1)
d3 <- dcomplnorm(0.4, "llogis", 0.5, 0.4, shape = 0.5, scale = 0.5)
segments(0.4, 0, 0.4, d3, col = "blue", lty = 2)
legend(1.5, 0.5, legend =
    c(expression(paste(sigma==0.4, ",", theta==0.5, ",", shape==1, ",", scale==0.8)),
      expression(paste(sigma==0.3, ",", theta==0.2, ",", shape==0.2, ",", scale==0.5)),
      expression(paste(sigma==0.5, ",", theta==0.4, ",", shape==0.5, ",", scale==0.8))),
      col = c("black", "red", "blue"), lty = 1)

curve(pcomplnorm(x, "llogis", 0.4, 0.5, shape = 1, scale = 0.8), xlim = c(0, 5),
      ylim = c(0, 1), xlab =  "x", ylab = "F(x)", n = 250, col = "black", lty = 1)
d1 <- pcomplnorm(0.5, "llogis", 0.4, 0.5, shape = 1, scale = 0.8)
segments(0.5, 0, 0.5, d1, col = "black", lty = 2)
curve(pcomplnorm(x, "llogis", 0.3, 0.2, shape = 0.2, scale = 0.5), add = TRUE,
      col = "red", lty = 1)
d2 <- pcomplnorm(0.2, "llogis", 0.3, 0.2, shape = 0.2, scale = 0.5)
segments(0.2, 0, 0.2, d2, col = "red", lty = 2)
curve(pcomplnorm(x, "llogis", 0.5, 0.4, shape = 0.5, scale = 0.5), add = TRUE,
      col = "blue", lty = 1)
d3 <- pcomplnorm(0.4, "llogis", 0.5, 0.4, shape = 0.5, scale = 0.5)
segments(0.4, 0, 0.4, d3, col = "blue", lty = 2)
```

Figure 1 shows the pdfs and cdfs of the composite lognormal-loglogistic distribution. The parameters, σ and θ, the scale parameter of the lognormal distribution and the cutoff point, respectively, are as defined in Section "Properties". The parameters, shape and scale, are the shape and scale parameters of the loglogistic distribution, as defined in the R base package. The vertical lines in the figures correspond to the values for θ, the cutoff point.

Illustration 2

The illustration presented here fits the composite lognormal-Fréchet distribution to the Danish fire insurance data by the method of maximum likelihood, see (9). The data were obtained from the R package **SMPracticals** (Davison, 2012).

```
x <- danish[1:2492]
nlm(function(p) {
        -sum(dcomplnorm(x, "frechet", sigma = exp(p[1]),
             theta = exp(p[2]), scale = exp(p[3]), shape = exp(p[4]), log = TRUE))
    }, p = c(0, 0, 0, 0))
```

The output will be

```
$minimum
[1] 3859.293

$estimate
[1] -1.7178497  0.1176224 -0.2872701  0.4130180

$gradient
[1] -0.0008301586  0.0031436684  0.0001900844 -0.0004574758

$code
[1] 1

$iterations
[1] 23
```

The number 2492 refers to the actual length of the Danish data set. This output shows that the maximized log-likelihood is -3859.293, the estimated σ is $\exp(-1.7178497) = 0.1794516$, the estimated θ is $\exp(0.1176224) = 1.124819$, the estimated scale parameter of the Fréchet distribution

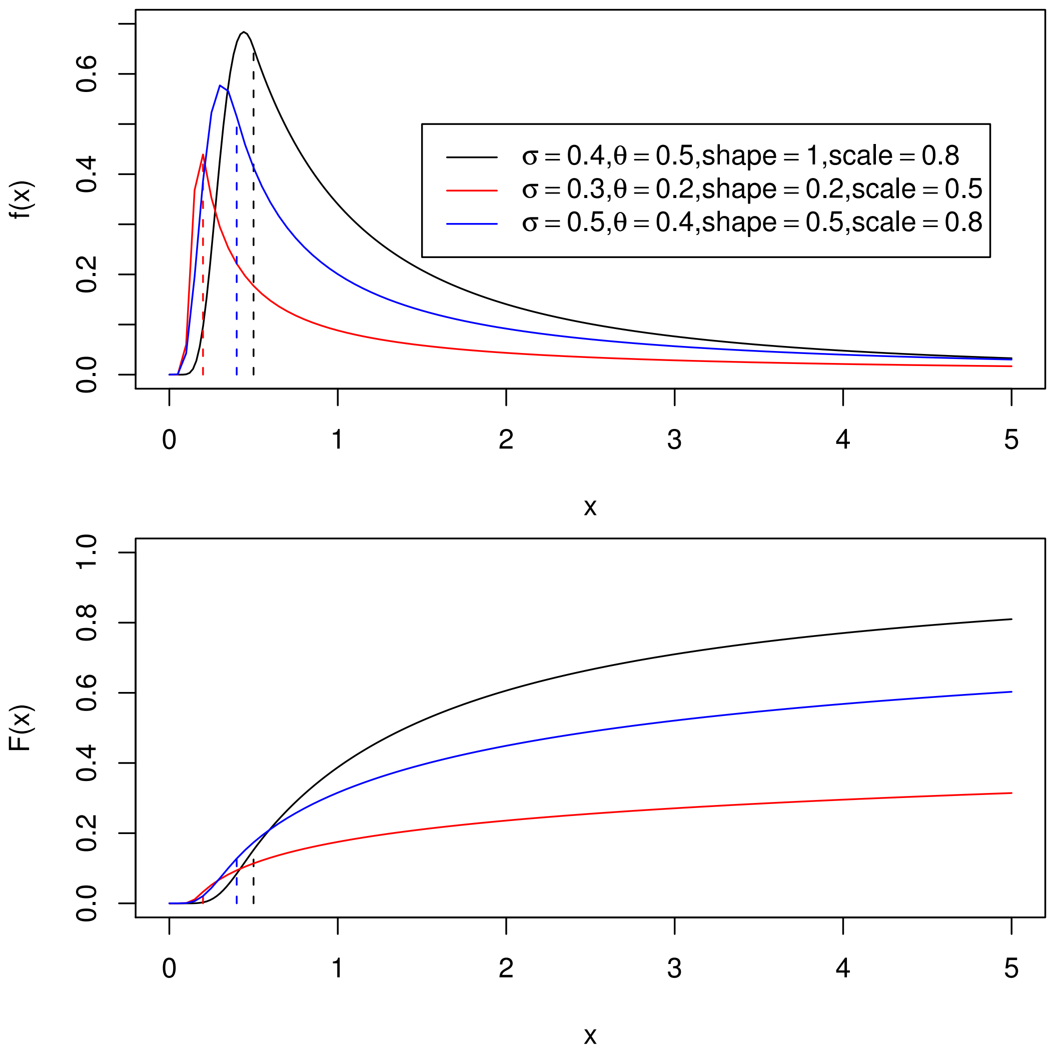


Figure 1: Pdfs (top) and cdfs (bottom) of the composite lognormal-loglogistic distribution. The top legend applies to both plots.

is $\exp(-0.2872701) = 0.750309$, and the estimated shape parameter of the Fréchet distribution is $\exp(0.4130180) = 1.511372$. The standard errors of these estimates can be computed by adding `hessian = TRUE` to the `nlm` command.

We have chosen the composite lognormal-Fréchet distribution for simplicity of illustration. The purpose of this illustration was not to find the "best fitting" model for the Danish data. Of course, many other composite lognormal distributions can be expected to give better fits than the composite lognormal-Fréchet distribution. Nadarajah and Bakar (2012) modeled the Danish data using a large class of composite lognormal distributions, including the composite lognormal-Burr, the composite lognormal-inverse Burr, the composite lognormal-*F*, the composite lognormal-Fréchet, the composite lognormal-generalized Pareto, the composite lognormal-inverse Pareto, and the composite lognormal-loglogistic distributions. The composite lognormal-Burr distribution was shown to give the best fit.

In Illustration 2, we used `nlm` for optimization of the likelihood function. R has many other model fitting routines like `fitdistr` from the package **MASS** (Venables and Ripley, 2002), `fitdist` from the package **fitdistrplus** (Delignette-Muller et al., 2013), `mle` from the package **stats4**, and `MLEstimator`, `MDEstimator` from the package **distrMod** (Kohl and Ruckdeschel, 2013). For instance, `fitdist` can be used to estimate the parameters of the composite lognormal-Fréchet distribution as follows:

```
dclnormf <- function(x, logsigma, logtheta, logscale, logshape) {
    dcomplnorm(x, spec = "frechet", sigma = exp(logsigma), theta = exp(logtheta),
               scale = exp(logscale), shape = exp(logshape))
}
pclnormf <- function(q, logsigma, logtheta, logscale, logshape) {
    pcomplnorm(q, spec = "frechet", sigma = exp(logsigma), theta = exp(logtheta),
               scale = exp(logscale), shape = exp(logshape))
}
qclnormf <- function(p, logsigma, logtheta, logscale, logshape) {
    qcomplnorm(p, spec = "frechet", sigma = exp(logsigma), theta = exp(logtheta),
               scale = exp(logscale), shape = exp(logshape))
}
fitdist(danish[1:2492], "clnormf", start = list(logsigma = -1.718,
        logtheta = 0.118, logscale = -0.287, logshape = 0.413))
```

The output will be

```
Fitting of the distribution ' clnormf ' by maximum likelihood
Parameters:
           estimate Std. Error
logsigma -1.7180280 0.05737326
logtheta  0.1176365 0.02413085
logscale -0.2877565 0.16564834
logshape  0.4130318 0.03704017
```

Also mle can be used to estimate the parameters of the composite lognormal-Fréchet distribution as follows:

```
nllh <- function(p1, p2, p3, p4) {
    -sum(dcomplnorm(danish[1:2492], spec = "frechet",
         sigma = exp(p1), theta = exp(p2), scale = exp(p3), shape = exp(p4), log = TRUE))
}
mle(nllh, start = list(p1 = -1.718, p2 = 0.118, p3 = -0.287, p4 = 0.413))
```

The output will be

```
Call:
mle(minuslogl = nllh, start = list(p1 = -1.718, p2 = 0.118, p3 = -0.287,
    p4 = 0.413))

Coefficients:
        p1         p2         p3         p4
-1.7178735  0.1176081 -0.2870383  0.4130586
```

Conclusions

We have developed a new R package for computing quantities of interest for *any* composite lognormal distribution. The computed quantities include the pdf, cdf, qf, and random numbers. Although the package is specifically designed for composite lognormal distributions, it can be easily altered for any other composite distribution.

Acknowledgments

The authors would like to thank the Editor and the two referees for careful reading and for their comments which greatly improved the paper.

Bibliography

S. Bakar. *Some Contributions to Actuarial Parametric Modeling*. PhD thesis, University of Manchester, UK, 2012. [p99]

P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. *ACM SIGMETRICS Performance Evaluation Review*, 26:151–160, 1998. [p98]

C. Bolancé, M. Guillén, and J. Nielsen. Transformation kernel estimation of insurance claim cost distributions. In *Mathematical and Statistical Methods for Actuarial Sciences and Finance*, pages 43–51, 2010. [p98]

K. Cooray and M. Ananda. Modeling actuarial data with a composite lognormal-Pareto model. *Scandinavian Actuarial Journal*, pages 321–334, 2005. [p98]

A. Davison. *SMPracticals: Practicals for Use with Davison (2003) Statistical Models*, 2012. URL `http://CRAN.R-project.org/package=SMPracticals`. R package version 1.4-1. [p101]

M. Delignette-Muller, R. Pouillot, J.-B. Denis, and C. Dutang. *fitdistrplus: Help to Fit of a Parametric Distribution to Non-Censored or Censored Data*, 2013. URL `http://cran.r-project.org/web/packages/fitdistrplus/fitdistrplus.pdf`. [p102]

S. Dubey, B. Ponnu, and S. Arkatkar. Time gap modeling using mixture distributions under mixed traffic conditions. *Journal of Transportation Systems Engineering and Information Technology*, 13:91–98, 2013. [p98]

I. Eliazar and M. Cohen. A Langevin approach to the log-Gauss-Pareto composite statistical structure. *Physica A: Statistical Mechanics and Its Applications*, 391:5598–5610, 2012. [p98]

M. Fackler. Inflation and excess insurance, 2010. ASTIN / AFIR Colloquium, Madrid. [p98]

J. Gibbons and S. Mylroie. Estimation of impurity profiles in ionimplanted amorphous targets using joined half-Gaussian distributions. *Applied Physics Letters*, 22:568–569, 1973. [p98]

C. Gillespie. *Fitting Heavy Tailed Distributions: The poweRlaw Package*, 2013. URL `http://cran.r-project.org/web/packages/poweRlaw/index.html`. [p100]

M. Kohl and P. Ruckdeschel. *distrMod: Object Oriented Implementation of Probability Models*, 2013. URL `http://cran.r-project.org/web/packages/distrMod/index.html`. [p102]

K.-S. Lindblad. How big is large? A study of the limit for large insurance claims in case reserves. Master's thesis, KTH/Matematisk statistik, 2011. [p98]

S. Nadarajah and S. Bakar. New composite models for the Danish fire insurance data. *Scandinavian Actuarial Journal*, 2012. DOI: 10.1080/03461238.2012.695748. [p98, 99, 102]

M. Pigeon and M. Denuit. Composite lognormal-Pareto model with random threshold. *Scandinavian Actuarial Journal*, pages 177–192, 2011. [p98]

D. Scollnik. On composite lognormal-Pareto models. *Scandinavian Actuarial Journal*, pages 20–33, 2007. [p98]

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. URL `http://www.stats.ox.ac.uk/pub/MASS4`. ISBN 0-387-95457-0. [p102]

S. Nadarajah
School of Mathematics
University of Manchester
Manchester M13 9PL, UK `mbbsssn2@manchester.ac.uk`

S. A. A. Bakar
Institute of Mathematical Sciences
University of Malaya
50603 Kuala Lumpur, Malaysia `saab@um.edu.my`

lfe: Linear Group Fixed Effects

by Simen Gaure

Abstract Linear models with fixed effects and many dummy variables are common in some fields. Such models are straightforward to estimate unless the factors have *too many* levels. The R package **lfe** solves this problem by implementing a generalization of the **within transformation** to multiple factors, tailored for large problems.

Introduction

A typical linear model looks like:

```
> y ~ x1+x2+x3 + f1+f2+f3
```

where `f1,f2,f3` are arbitrary factors, and `x1,x2,x3` are other covariates. Coefficients may easily be estimated by `lm()`:

```
> lm(y ~ x1+x2+x3 + f1+f2+f3)
```

However, in some applications, in particular in econometrics, the factors have too many levels and are still needed as fixed effects, as in Abowd et al. (1999). The use of fixed effects as opposed to random effects is typically necessitated by the possibility that the factors and the other regressors are correlated. They study log wage (`logwage`) as an outcome, where fixed effects for individuals (`id`) and firms (`firm`) are included, with some ordinary covariates (exemplified by `x`), i.e. a model of the type:

```
> logwage ~ x + id + firm
```

where `id` is a factor with one level for each employee, and `firm` is a factor with one level for each firm. Employees change firm now and then, so it is not a nested model. The model is used to account for arbitrarily distributed time constant individual and firm hetereogeneity, and is used to study the correlation between the firm effect and the employee effect. There are also similar cases with 3 factors, e.g. in Torres et al. (2013), where job title is also considered. They have a dataset of 27 million observations, with 5.5 million, 568,000, and 96,000 levels in the three factors. In other applications the factors are primarily used as controls, as in Markussen and Røed (2012), where a variety of models are used, controlling for various combinations of interactions between factors. These datasets are typically sourced from large public registries, and can contain tens of millions of observations, with hundreds of thousands or millions of factor levels. This far exceeds the capabilities of `lm()`, and can also be too demanding for the sparse methods in package **Matrix** (Bates and Maechler, 2013).

When estimating such models, the case with a single factor is special. It can be estimated by using the within groups transformation, where the mean of the groups are subtracted from the covariates, resulting in a system without the factor, via the Frisch-Waugh-Lovell theorem. See e.g. Wooldridge (2002, Section 10.5). The coefficients for the factor levels can easily be recovered as the group means of the residuals. This estimation method can be found in package **plm** (Croissant and Millo, 2008). The method can also be used with more than one factor, by using the within transformation to project out the factor with the highest number of levels, coding the others as dummy variables. However, if all of the factors have many levels this can still result in a too large system which is non-sparse. Moreover, having such sets of dummies in datasets which are not balanced may lead to non-trivial identification problems. We will illustrate how to use the package **lfe** (Gaure, 2013b) to solve some of these problems. For clarity we construct quite small datasets by drawing random covariates, rather than to refer to large real datasets which are not publicly available.

Enter lfe

The package **lfe** is designed to handle the above estimation problem. It also contains some methods for solving identification problems, though not completely in all cases. Since **lfe** handles quite ordinary linear models which conceptually could be handled by `lm()`, we do not go into detail about the feasibility of fixed effect linear models as such, only the problems which are more or less peculiar to models with a large number of dummies.

The short story is that coefficients for `x1`, `x2`, and `x3` in the above model can be estimated by:

```
> library(lfe)
> est <- felm(y ~ x1+x2+x3 + G(f1)+G(f2)+G(f3))
```

whereas the coefficients for the factor levels of `f1`, `f2`, and `f3` can be retrieved by:

```
> alpha <- getfe(est)
```

A longer story follows, the theoretical part of which is mainly based on Gaure (2013a).

We write our model in matrix form

$$y = X\beta + D\alpha + \epsilon, \tag{1}$$

where D is a matrix of dummies coding the factor levels, X is a matrix with the other covariates, y is the response vector, and ϵ is a normally distributed error term. Performing an ordinary least squares regression (OLS) on this system yields the OLS estimates $\hat{\beta}$ for the X covariates and $\hat{\alpha}$ for the factor levels. The Frisch-Waugh-Lovell theorem states that if P is the projection onto the orthogonal complement of the range of D, then the projected system

$$Py = PX\beta + P\epsilon,$$

yields the same $\hat{\beta}$ when estimated with OLS. Moreover, the matrix $(X^tPX)^{-1}$ which is used to find the covariance matrix of $\hat{\beta}$, is identical to the $\hat{\beta}$-part of the corresponding matrix in the full system (1). Similarly, the residuals are identical. The projected system does not contain the dummies for the factor levels, and may therefore be manageable with conventional methods.

Unfortunately, P is an $n \times n$ matrix, where n is the number of observations, so it is impractical to compute P when $n \approx 10^7$. However, it is easier to compute Px for a vector x, i.e. y and the columns of X. In the special case when D encodes a single factor, the transformation $x \mapsto Px$ is the within transformation, i.e. centring of the groups on their means. It is shown in Gaure (2013a) that when there is more than one factor, say $e > 1$ factors, we may consider the centring transformation for each of them, a projection P_i for each $i = 1 \ldots e$, and compute Px as

$$Px = \lim_{m\to\infty} \left((P_1 P_2 \cdots P_e)^m \, x \right).$$

This approach is known as the *method of alternating projections* and is based on a result by Halperin (1962, Theorem 1). The procedure can easily be implemented with an R function taking as input a vector x and the list of factors `flist`:

```
> demean <- function(x, flist) {
+   cx <- x; oldx <- x - 1
+   while(sqrt(sum((cx - oldx) ^ 2)) >= 1e-8) {
+     oldx <- cx
+     for(f in flist) cx <- cx - ave(cx, f)
+   }
+   return(cx)
+ }
```

This algorithm was also arrived at by Guimarães and Portugal (2010, p. 637) as a technical simplification of their iterated estimation approach to the same problem.

For efficiency reasons, this linear transformation has been written in C, made threaded to centre vectors in parallel, and is available as the function `demeanlist()` in **lfe**, though the function `felm()` wraps it in an `lm()` like function.

We create a simple example to illustrate the usage. We have 100,000 observations of a covariate `x`, and two factors `f1`, `f2`, each with 10,000 randomly drawn levels. We create an outcome variable `y` and estimate the `x` coefficient. The `G()` syntax is used to specify which factors should be projected out of the system, a similar syntax as for the `Error()` term in `aov()`. The `G()` is not an R function in itself, though it translates to `as.factor()` inside `felm()` after the `G()` terms have been removed from the model for special handling.

```
> library(lfe)
> set.seed(42)
> x <- rnorm(100000)
> f1 <- sample(10000, length(x), replace=TRUE)
> f2 <- sample(10000, length(x), replace=TRUE)
> y <- 2.13*x + cos(f1) + log(f2+1) + rnorm(length(x), sd=0.5)
> est <- felm(y ~ x + G(f1) + G(f2))
> summary(est)

Call:
   felm(formula = y ~ x + G(f1) + G(f2))

Residuals:
       Min         1Q     Median         3Q        Max
-1.9531308 -0.3018539 -0.0003573  0.3007738  2.2052754

Coefficients:
  Estimate Std. Error t value Pr(>|t|)
x 2.130889   0.001768    1205   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.5013 on 80000 degrees of freedom
Multiple R-squared: 0.9683   Adjusted R-squared: 0.9603
F-statistic: 122.1 on 19999 and 80000 DF, p-value: < 2.2e-16
```

The result of `felm()` is a '`felm`' object which is quite similar to an '`lm`' object. However, it is not fully compatible with it, so some methods for '`lm`' objects may work on a '`felm`' object, others may not. In an earlier version, the '`felm`' object inherited from the class '`lm`', but there are important differences in the structure, so this inheritance has been removed. In particular there is no qr-decomposition in the '`felm`' object, so methods for '`lm`' objects which depend on the qr-decomposition, such as `anova()`, can not be used. Also, the '`felm`'-object does not contain a copy of the dataset. This has been removed to conserve memory for very large datasets. Simple extractors like `coef()` and `residuals()` work. Extracting the covariance matrix with `vcov()` also works, via a separate S3 method, albeit only for the $\hat{\beta}$s. Also, a `summary()` S3-method, and an accompanying `print()` method have been included. It is possible to try '`lm`'-methods explicitly as in: `getS3method('vcov','lm')(est)`, though the author does not guarantee any success with this approach.

The careful reader has noticed that the behaviour of `summary()` on a '`felm`' object with respect to degrees of freedom and R^2 is the same as that of on an '`lm`' object when including an intercept. There is no explicit intercept in the result of `felm()`, but the factor structure includes one implicitly.

The coefficients for the factor levels

If the factors are used as controls only, the above procedure is all we have to worry about, but if we also need the group coefficients $\hat{\alpha}$, what econometricians often refer to as *the* fixed effects, we can solve the equation

$$D\hat{\alpha} = (I - P)(y - X\hat{\beta}), \tag{2}$$

for $\hat{\alpha}$. The right hand side is easily computed when we have $\hat{\beta}$, PX and Py. The equation is solved by the Kaczmarz method (Kaczmarz, 1937), as described in Gaure (2013a). The method is available as a function `kaczmarz()`, but the wrapper `getfe()` is more useful. A solution of equation (2) is not unique, the matrix D dummy-encodes *all* the factor levels, hence there will be multicollinearities (unless there is only one factor), both the obvious ones, but there can also be spurious ones. The multicollinearities are resolved by applying an estimable function. **lfe** contains a function `efactory()` for creating estimable functions, but users can also supply their own.

`getfe()` returns a '`data.frame`' with some information in addition to the coefficients. The reason for this is that the intended use of **lfe** is for factors with so many levels that they probably anyway must be analyzed according to the researcher's needs prior to being presented. We continue the example:

```
> alpha <- getfe(est)
> nrow(alpha)

[1] 20000

> alpha[9998:10003,]

                effect obs comp fe   idx
f1.9998  -0.2431720   9    1 f1  9998
f1.9999  -0.9733257   5    1 f1  9999
f1.10000 -0.8456289   9    1 f1 10000
f2.1      0.4800013   9    1 f2     1
f2.2      1.4868744  14    1 f2     2
f2.3      1.5002583  11    1 f2     3
```

The '`obs`' column is the number of observations of this level. The '`fe`' and '`idx`' columns are `factor`s containing the same information as the row name, but split into the name of the factor and the *level* for convenience. The '`comp`' column is important in that it is used for identification purposes, and here is a main deviation from how `lm()` treats identification in the presence of factors. The default method when using lm() is taken from the global 'contrasts' option which defaults to using treatment contrasts for each factor. It introduces a reference level for each factor, this is equivalent to forcing the coefficient for the reference level to zero, and the other coefficients are only meaningful when compared to this zero coefficient. It is also possible to specify different constraints for the lm() coefficients, such as forcing the sum of the coefficients to zero, or an arbitrary one via the 'contrasts' option, or the 'contrasts' argument to lm(), but typically a single constraint is used for each factor.

Identification when the factors have many levels can be a more complicated matter. The standard example in the econometrics literature is the one found in Abowd et al. (1999), elaborated in Abowd et al. (2002). In this case there are two factors, one for employees and one for firms. It may happen that one set of employees move between one set of firms, whereas another disjoint set of employees move between some other firms. There are no movements between these *mobility groups,* hence coefficients from different groups can not be compared. A useful construction for analysis of this problem is the undirected, bipartite graph which has the factor levels as vertices, and an edge between levels that occur in the same observation. The mobility groups are then the connected components of this graph. It is shown in Abowd et al. (2002, Appendix 1) that for identification of the coefficients, it is sufficient to introduce a single reference level in each of the disjoint mobility groups, either a firm or an employee. This was previously established by Eccleston and Hedayat (1974) in a different context, and there is another argument in Gaure (2013a) using spectral graph theory. The 'comp' column in the return value from getfe() when using estimable functions from efactory() is a factor enumerating these groups, i.e. the connected components. efactory() chooses by default the level with the highest number of observations as a reference level, and sets the coefficient to 0. When interpreting the coefficients, they should never be compared between the components; a coefficient is only meaningful relative to the reference level in the component it belongs to. For this reason, one may choose to restrict attention to the largest component only, the one with comp == 1, if this is feasible for the problem at hand.

To the author's knowledge, the identification problem when there are more than two factors has not been solved in general. efactory()'s behaviour with more than two factors is to assume that the connected components of the two first factors are sufficient for identification, and a single reference is used in each of the remaining factors. **lfe** contains a probabilistic test for estimability in this case, the one described in Gaure (2013a, Remark 6.2), and getfe() will issue a warning if it finds an identification problem. The test is also available as the function is.estimable(). In theory, the test can fail only on a set of measure zero. Specifically, the test uses the fact that an estimable function must evaluate to the same value on all solutions of equation (2). Different solutions can be obtained by running the Kaczmarz algorithm with different initial vectors. By starting with the zero vector we obtain the solution with least Euclidean norm. The test works by comparing the value of the candidate estimable function on the least norm solution and a solution obtained by drawing the initial vector η at random. The test will only fail to find an identification problem if η happens to lie in a particular, but unknown, subspace of positive codimension, i.e. in a set of Lebesgue measure zero. However, due to numerical inaccuracies, finite arithmetic, and bad luck, the test may in practice fail to find an identification problem even if there is one, with some very small positive probability. It does however not report identification problems when there are none. If there are identification problems which are unaccounted for, then some, or all of the resulting coefficients are non-estimable, i.e. they are meaningless.

Sometimes, an identification problem can be alleviated by changing the order of the G() terms in the formula supplied to felm(). To illustrate, consider a situation where we add a third factor to the example in the Introduction, nkids, the number of an individual's offspring, with only 5 levels. If our formula contains G(firm) + G(nkids) + G(id), it is likely that the graph associated with the two first factors, firm and nkids, is connected, and there can be many mobility groups which are unaccounted for. getfe() issues a warning, the returned coefficients are non-estimable; there is no correct indication of which mobility groups the coefficients belong to, and therefore, we do not know which coefficients can be meaningfully compared. If we change the model specification to G(id) + G(firm) + G(nkids), or G(id) + G(firm) + nkids, the mobility groups are found, and accounted for.

Another approach to the identification problem is found in Carneiro et al. (2012). They restrict attention to a subset of the dataset in which, by construction, all elementary contrasts, i.e. differences between coefficients, are estimable, as described by Weeks and Williams (1964). Such a partitioning of the dataset can be found by constructing a graph with the observations as vertices, and with an edge between two observations if they differ in at most a single factor. The partitions of the dataset correspond to connected components of the graph. They can be found by the function compfactor(...,WW=TRUE), which returns a factor enumerating the partitions. It can be used to select a subset of the dataset prior to calling felm(). When there are only two factors, this partitioning structure coincides with the mobility groups discussed above. In some datasets, like the one in Torres et al. (2013), the largest partition comprises almost the entire dataset, but if it does not, such a selection may introduce bias in the coefficients. An extreme example:

```
> set.seed(42)
> f1 <- factor(sample(50, 1000, replace=TRUE))
> f2 <- factor(sample(50, 1000, replace=TRUE))
> f3 <- factor(sample(50, 1000, replace=TRUE))
> ww <- compfactor(list(f1,f2,f3), WW=TRUE)
> head(table(ww))

ww
 1  2  3  4  5  6
29 20 19 16 14 14
```

The largest partition has 29 observations, fewer than the number of levels, even though far more estimable functions exist. This can be seen by computing the rank deficiency of the matrix D with the function rankMatrix() from package **Matrix**. Indeed, it is sufficient with two references in the 3 factors:

```
> D <- t(do.call('rBind', lapply(list(f1,f2,f3), as, 'sparseMatrix')))
> ncol(D) - as.integer(rankMatrix(D))

[1] 2
```

The standard method of efactory(), to analyze the connected components of the first two factors only, works well in this particular case. It will find a reference among the two first factors, and a reference in the last, and differences between coefficients within each factor will be estimable:

```
> x <- rnorm(1000)
> y <- 3.14*x + log(1:50)[f1] + cos(1:50)[f2] + exp(sqrt(1:50))[f3] + rnorm(1000, sd=0.5)
> est <- felm(y ~ x + G(f1) + G(f2) + G(f3))
> coef(est)

       x
3.139781

> is.estimable(efactory(est), est$fe)

[1] TRUE
```

This can happen because the construction in Weeks and Williams (1964, Theorem 1) does not find a maximal subset, only a subset which is guaranteed to have the desired property.

Yet another algorithm for finding estimable functions is described by Godolphin and Godolphin (2001), but **lfe** does not implement it.

Specifying an estimable function

The function efactory() in **lfe** is by default called by getfe() to create an estimable function for a factor structure. The default is to use reference levels as described above, some other estimable functions are also available. However, researchers may have other needs, so it is possible to supply a user written estimable function to getfe(). We describe this interface with an example, for a small dataset. The function takes as an argument a vector of length the sum of the number of levels of the factors, i.e. a solution to equation (2), applies an estimable function of choice, and returns the result. It is not necessary to include a full set of estimable functions; if so desired, our function may return just a single difference between two specific factor levels, or even some nonlinear transformation thereof. It should, however, evaluate to the same value on all the different solutions of equation (2).

We make a small dataset with 100 observations, a covariate x and 3 factors with a handful of levels.

```
> set.seed(42)
> x <- rnorm(100)
> f1 <- factor(sample(4, 100, replace=TRUE))
> f2 <- factor(sample(5, 100, replace=TRUE))
> f3 <- factor(sample(6, 100, replace=TRUE))
> e1 <- sin(1:4)[f1] + 0.02*((1:5)^2)[f2]  + 0.17*((1:6)^3)[f3] + rnorm(100)
> y <-  2.5*x + (e1-mean(e1))
> est <- felm(y ~ x + G(f1) + G(f2) + G(f3))
```

Then we create an estimable function which is the same as the one provided by lm() when using treatment contrasts. The addnames argument is there because in the event that the estimable function is used in bootstrapping, adding names to very long vectors would only contribute to exercising R's memory management; hence names should only be added when required. It is also possible to add an `attribute` called `'extra'`, which is a named list of vectors of the same length as the names, for adding additional information to the result of `getfe()`. This `attribute` is added by the estimable functions returned by `efactory()` to provide the `'obs'`, `'fe'`, `'comp'` and `'idx'` columns, but the content is not used for anything inside **lfe**.

```
> ef <- function(gamma, addnames) {
+   ref1 <- gamma[1]  # first level of f1
+   ref2 <- gamma[5]  # first level of f2
+   ref3 <- gamma[10] # first level of f3
+   # put the intercept in the first coordinate
+   icpt <- ref1 + ref2 + ref3
+   # subtract the references for each factor
+   # unlike the efactory() functions, we omit the zero coefficients.
+   result <- c(icpt, gamma[2:4]-ref1, gamma[6:9]-ref2, gamma[11:15]-ref3)
+   if(addnames) {
+     names(result) <- c('(Intercept)',
+                        paste('f1', levels(f1)[2:4], sep=''),
+                        paste('f2', levels(f2)[2:5], sep=''),
+                        paste('f3', levels(f3)[2:6], sep=''))
+     attr(result, 'extra') <- list(fe=factor(
+                                     c('icpt', rep('f1',3),
+                                       rep('f2',4), rep('f3',5))),
+                                   idx=factor(c(1, 2:4, 2:5, 2:6)))
+   }
+   result
+ }
```

We now check that our function is estimable:

```
> is.estimable(ef, list(f1,f2,f3))
```

```
[1] TRUE
```

The estimable function is supplied to `getfe()` via the argument `ef`. In this example we also request bootstrapped standard errors with the arguments `se` and `bN`, we elaborate on this in the next section.

```
> getfe(est, ef=ef, se=TRUE, bN=1000)
```

```
                 effect   fe idx        se
(Intercept) -10.9016327 icpt   1 0.3077378
f12          -0.1265879   f1   2 0.2356162
f13          -0.7541019   f1   3 0.2896058
f14          -1.7409436   f1   4 0.2776542
f22           0.4611797   f2   2 0.3012931
f23           0.6852553   f2   3 0.2898361
f24           0.8467309   f2   4 0.3232411
f25           0.5886517   f2   5 0.2841049
f32           1.0898551   f3   2 0.3364884
f33           4.3490898   f3   3 0.3058420
f34          10.7505266   f3   4 0.3377505
f35          21.3832700   f3   5 0.3649107
f36          36.7369397   f3   6 0.3059049
```

`getfe()` performs no automatic addition of columns like `'idx'` or `'comp'`, they have to be provided by the estimable function. The reason being that `getfe()` does not know the structure imposed by the user written function. If bootstrapped standard errors are requested via the `se` argument, the `'se'` column is added also for user written functions. If you are certain that your function is estimable, you can add an attribute to it, `attr(ef,'verified') <- TRUE`, this causes `getfe()` to skip the estimability test, and may save significant amounts of time with datasets for which convergence is slow. The functions produced by `efactory()` for one and two factors have this attribute set.

Standard errors

The standard errors for $\hat{\beta}$ are easily computed, and are identical to the the standard errors from `lm()` if we were to run it with all the dummies. However, there is a minor difference. The degrees of freedom depends on the rank of the matrix D, or D^tD, or the trace of P. The rank is easily computed for the cases with one or two factors, but requires a much more time consuming approach for 3 or more factors. The default approach of `felm()` is to assume that the column rank deficiency of D is $e-1$ when there are $e > 2$ factors. This may result in a too high value for the rank, hence a too low value for the degrees of freedom, which will yield too high standard errors. In most real cases the author has witnessed, the error is negligible. `felm()` has an argument `exactDOF` which may be set to `TRUE` to activate a more accurate computation of the rank. It does a sparse pivoted Cholesky factorization of $D^tD + \epsilon I$ for some small ϵ and finds the rank deficiency by counting small pivots. This is not a perfect method, as noted by Higham (1990), but it seems to work well for the matrices the author has encountered, and it is much faster than `rankMatrix()` from package **Matrix**. To use `rankMatrix()` instead, one may specify `exactDOF='rM'`. If, for some reason, one happens to know the degrees of freedom in advance, they can be specified as a numeric, like `exactDOF=12536819`.

The standard errors for $\hat{\alpha}$, the coefficients for the factor levels, can be estimated by bootstrapping. This requires resampling the residuals with replacement, and do the whole estimation over and over again, an operation which can be very time consuming. It can be requested, as in the above example, by the `se` argument to `getfe()`, the number of samples can be specified in the `bN` argument. The common practice of resampling observations rather than residuals is not useful for this kind of model, it results in a different factor structure, hence possibly a different set of identified coefficients in each sample.

Instrumental variables

lfe supports instrumental variables estimation through 2SLS, i.e. two step OLS (Wooldridge, 2002, Chapter 5).

Here is an example. We first create some covariates:

```
> set.seed(276709)
> x <- rnorm(10000)
> x2 <- rnorm(length(x))
> x3 <- rnorm(length(x))
```

Then we create some factors, and their effects:

```
> id <- factor(sample(2000, length(x), replace=TRUE))
> firm <- factor(sample(1300, length(x), replace=TRUE))
> id.eff <- rnorm(nlevels(id))
> firm.eff <- rnorm(nlevels(firm))
```

and a normally distributed error term `u`, and an outcome `y`:

```
> u <- rnorm(length(x))
> y <- x + 0.5*x2 + id.eff[id] + firm.eff[firm] + u
```

We create a covariate `Q` which is correlated with the other covariates, the instrument `x3`, and the error term, and add it to the outcome:

```
> Q <- 0.3*x3 + x + 0.2*x2 + 0.5*id.eff[id] + 0.7*u + rnorm(length(x), sd=0.3)
> y <- y + 0.9*Q
```

Estimation is now carried out by specifying the instrumented variable equation with the `iv` argument of `felm`. Only the instrument variable, in this case `x3`, is needed, the other covariates are added by `felm()`.

```
> ivest <- felm(y ~ x + x2 + G(id) + G(firm) + Q, iv=Q ~ x3)
> summary(ivest)

Call:
   felm(formula = y ~ x + x2 + G(id) + G(firm) + Q, iv = Q ~ x3)

Residuals:
      Min        1Q    Median        3Q       Max
-6.036142 -0.903260  0.000759  0.913758  4.971614

Coefficients:
         Estimate Std. Error t value Pr(>|t|)
x         0.94963    0.03975   23.89   <2e-16 ***
```

```
x2          0.49567    0.01449    34.20   <2e-16 ***
`Q(fit)`    0.94297    0.03816    24.71   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.668 on 6717 degrees of freedom
Multiple R-squared: 0.8112   Adjusted R-squared: 0.7189
F-statistic: 8.791 on 3282 and 6717 DF, p-value: < 2.2e-16
```

`felm()` supports more than one instrumented variable, in this case the `iv` argument must be a `list` of `formulas`, one for each instrumented variable, with all the instruments on the right hand side.

The author has been made aware that both the `G()` syntax and the handling of instrumental variables equations could be done in a smoother fashion with multi-part `formulas`. Such a syntax is not presently available in **lfe**.

Estimation time

It is a fact of life that both `felm()` and `getfe()` may take quite a while to complete. What is more unfortunate is that time to completion does not only depend on the size of the dataset, but also on its structure. In particular, the dependencies between the factors, beyond the pure identification problems, have a huge impact. The rate of convergence for the general method of alternating projections has been analyzed by Aronszajn (1950); Kayalar and Weinert (1988); Gearhart and Koshy (1989); Deutsch and Hundal (1997); Bauschke et al. (2003); Badea et al. (2012), among others. Their results are in terms of the cosine c of generalized angles between the subspaces corresponding to the projections P_i. For two factors, the convergence rate in operator norm is known to be

$$\|(P_1P_2)^n - P\| = c^{2n-1}, \tag{3}$$

where $0 \leq c < 1$. The convergence is linear in the case with two factors, but the rate depends heavily on the structure of the factors. With 3 or more factors, the convergence rate depends on both the structure of the factors and their order in the iterations, in a quite complicated way (Deutsch and Hundal, 1997, Theorem 2.7). The theoretical convergence results are also valid for the Kaczmarz method used by `getfe()`, as this is a special case of the method of alternating projections (Deutsch and Hundal, 1997, Section 4). Though, for our Kaczmarz step the number of projections is the number of observations, not the number of factors.

We do not offer general results which relate intuitive properties of the factors to the convergence rate, but here are some small examples to illustrate the complexity.

In our first example the factors `f1` and `f2` are independent:

```
> set.seed(54)
> x <- rnorm(100000)
> f1 <- sample(10000, length(x), replace=TRUE)
> f2 <- sample(300, length(x), replace=TRUE)
> y <- x + cos(f1) + log(f2+1) + rnorm(length(x), sd=0.5)
```

We time the estimation:

```
> system.time(est <- felm(y ~ x + G(f1) + G(f2)))

   user  system elapsed
  2.420   0.008   1.955

> system.time(alpha <- getfe(est))

   user  system elapsed
  0.256   0.008   0.263
```

This is a quite fast example, in general it is the author's experience that datasets with this kind of full independence between the factors converge quite fast. Convergence is equally fast with 10,000 levels in the second factor as well.

But then we introduce a dependence which makes convergence slow. We let the second factor be closely related to the first, with some stochasticity, but we do not increase the number of levels. In this example, the second factor `f3` can only have 5 different values for each value of `f1`.

```
> f3 <- (f1 + sample(5, length(x), replace=TRUE)) %% 300
> y <- x + cos(f1) + log(f3+1) + rnorm(length(x), sd=0.5)
> system.time(est <- felm(y ~ x + G(f1) + G(f3)))

   user  system elapsed
 34.624   0.000  18.804

> system.time(alpha <- getfe(est))

   user  system elapsed
  3.868   0.008   3.880
```

Execution time increases by an order of magnitude, even though the size of the dataset is the same as before. In this example, with only 300 levels in the second factor, we may as well encode it as ordinary dummies, reducing our model to the classical within groups estimator, with 300 covariates:

```
> system.time(est <- felm(y ~ x + G(f1) + factor(f3)))

   user  system elapsed
 10.340   0.832   5.379

> length(coef(est))

[1] 300

> system.time(alpha <- getfe(est))

   user  system elapsed
  0.192   0.000   0.192
```

This is far from being the whole story. A "small" change to the second factor brings the execution time back down. We still have only 5 different values of the second factor f4 for each value of f1, but they are spread irregularly apart:

```
> f4 <- (f1 + sample(5, length(x), replace=TRUE)^3) %% 300
> y <- x + cos(f1) + log(f4+1) + rnorm(length(x), sd=0.5)
> system.time(est <- felm(y ~ x + G(f1) + G(f4)))

   user  system elapsed
  2.564   0.000   2.081

> system.time(alpha <- getfe(est))

   user  system elapsed
  0.240   0.004   0.244
```

To better appreciate the complexity, we create two more examples which are similar to the previous one, but the randomly drawn numbers are spread regularly apart. The first one results in slow convergence:

```
> f5 <- (f1 + sample(seq(1,197,49), length(x), replace=TRUE)) %% 300
> y <- x + cos(f1) + log(f5+1) + rnorm(length(x), sd=0.5)
> system.time(est <- felm(y ~ x + G(f1) + G(f5)))

   user  system elapsed
 32.636   0.000  17.368

> system.time(alpha <- getfe(est))

   user  system elapsed
  3.972   0.000   3.975
```

The second one converges fast:

```
> f6 <- (f1 + sample(seq(1,201,50), length(x), replace=TRUE)) %% 300
> y <- x + cos(f1) + log(f6+1) + rnorm(length(x), sd=0.5)
> system.time(est <- felm(y ~ x + G(f1) + G(f6)))

   user  system elapsed
  2.548   0.000   2.073
```

```
> system.time(alpha <- getfe(est))

   user  system elapsed 
  0.244   0.000   0.244 
```

By comparing the "user" and "elapsed" times for `felm()` in the above examples, it can be inferred that most of the time in the fast examples is spent in bookkeeping outside the centring, otherwise "user" time would be close to twice the "elapsed" time, since the actual centring of `y` and `x` is done in parallel. This means that the actual centring convergence rate differences are larger than the reported differences in execution time. For a careful analysis of centring times only, the centring process should be timed directly, as mentioned in Gaure (2013a, top of p. 17).

The author has not succeeded in finding general intuitive guidelines for the convergence rate. However, with two factors, a relevant concept seems to be the bipartite graph where the vertices are factor levels and there is an edge between levels which are observed together. It is the connected components of this graph which determines estimability. The graph can be analyzed with the tools in package **igraph** (Csardi and Nepusz, 2006) as follows:

```
> library(igraph)
> mkgraph <- function(flist) {
+   graph.adjacency(tcrossprod(do.call('rBind',
+                                    lapply(flist, as, 'sparseMatrix')))>0,
+                   'undirected', diag=FALSE)
+ }
```

We make a list of the associated graphs:

```
> glist <- lapply(list(f2,f3,f4,f5,f6),
+                function(f) mkgraph(lapply(list(f1,f), factor)))
```

The graphs, except for the last, are connected:

```
> sapply(glist, no.clusters)

[1]  1  1  1  1 50
```

The average degrees, i.e. the number of edges in the graph, show no convincing signs of being strongly related to the estimation time:

```
> sapply(glist, function(g) mean(degree(g)))

[1] 19.100301  8.404311  8.410719  8.400039  8.423925
```

A property of the graph which shows some promise of correlating with the convergence rate is the *diameter*. To get some intuituion of what this is, consider the example with firms and employees. The associated graph has as vertices the firms and employees, and there is an edge between a firm and an employee if the employee has worked for the firm. A *path* between e.g. two employees *Roy* and *Floyd* can be constructed by finding a firm that *Roy* has worked for, then another employee *Warshall* of this firm, then another firm for *Warshall*, zigzagging between firms and their employees until we reach the employee *Floyd*. For each pair of vertices (i.e. firms and employees) in a mobility group, there is a shortest path. The length of the longest of the shortest paths among all vertex pairs is the graph's diameter.

In these particular examples, it turns out that in the cases with fast convergence (the first, third, and fifth), the shortest paths between pairs of vertices are typically much shorter than in the slowly converging cases. We do not compute the diameter as a typical fast algorithm for this, the *Roy-Warshall-Floyd* algorithm, has complexity of order $O(n^3)$, where n is the number of vertices, i.e. 10300 in our example. Below, for simplicity, we exclude the last graph, it is disconnected, hence of infinite diameter, though the diameters of its (comparably small) components are relevant. A component of this graph would typically contain only about $300/50 = 6$ different levels of the second factor `f6`, so its diameter can't possibly be very large.

```
> for(gr in glist[1:4])
+   print(fivenum(shortest.paths(gr, v=sample(V(gr),10), to=sample(V(gr),10))))

[1]  2  2  4  4  4
[1]  2 20 39 62 76
[1]  2  4  6  6  8
[1]  2 18 40 58 76
```

Also, the many variants of factor structures in Markussen and Røed (2012) (described in Gaure (2013a)) and ongoing subsequent works, use factors which are interactions of other factors, in such a way that there are a few collinearities by design. This is not a conceptual problem when the factors are used as controls, but manually removing these known collinearities by merging some levels into a common "reference" level, yields performance improvements of up to two orders of magnitude (Markussen, 2013), a phenomenon which was *not* noted in Gaure (2013a). Incidentally, this also leads to shorter paths via the common reference level.

This suggests that, loosely speaking, datasets with six degrees of separation converge fast, whereas less well connected datasets converge slower. However, we cannot rule out other structural differences which may have an impact on the convergence rate; there is no such thing as "larger diameter, ceteris paribus" for graphs. It is possible that a diameter path could be used to construct a lower bound for the c in equation (3), but the author has not found a proof for such an assertion.

The convergence rate with more than two factors is more complicated, the theoretical results of Deutsch and Hundal (1997) are less complete, the rate is not a function of the cosines of *pairs* of subspaces, and even the order of the projections matters. In cases where only some of the factors have many levels, the remaining factors may be specified without `G()`, treating them as ordinary dummy-encoded covariates.

lfe utilizes two acceleration techniques for the alternating projections methods. In `demeanlist()`, which is used by `felm()` to centre the covariates, the line search method in Bauschke et al. (2003, Theorem 3.16) is used, even though their *Example 3.24* shows that in some special cases with more than two factors, this method is in fact slower. For the Kaczmarz method, random shuffling of the equations is used, as suggested by Gaure (2013a).

Parallelism

`felm()` is thread parallelized over the vectors to be centred, i.e. all variables in the model except those enclosed in `G()`. The number of processors is fetched with `getOptions('lfe.threads')` which is initialized upon loading of **lfe** from the environment variable `LFE_THREADS` or `OMP_NUM_THREADS`, or otherwise from an heuristic snatched from package **multicore** (Urbanek, 2011). It may of course be set manually after loading the package. There is no benefit from specifying more threads than there are variables to centre. `getfe()` is not parallelized as such, but bootstrapping with `getfe(...,se=TRUE)` is.

Elsewhere

lfe is similar in function, but not in method, to the Stata (StataCorp., 2013) modules A2reg (Ouazad, 2008) and felsdvreg (Cornelißen, 2008). It is the same algorithm as in the Stata module reg2hdfe (Guimarães, 2009), described by Guimarães and Portugal (2010) in terms of Gauss-Seidel iterations.

Summary

The package **lfe** contains methods for estimating ordinary least square models with multiple factors with too many levels for conventional methods like `lm()`. Such models may exhibit non-trivial identification problems. These are satisfactorily solved for the two factor case, and the package also includes some partial solutions for the case with more factors. Instrumental variable regression via the two step OLS is also supported. Convergence rate can be an issue for some datasets, and the present paper suggests some intuitive structural reasons for this.

The method employed by **lfe** is known as the *method of alternating projections*, a somewhat old and well studied method which to the author's knowledge has not been applied to the particular problem of linear regression with dummy variables before.

Acknowledgements

I wish to thank the associate editor and two anonymous reviewers for helpful comments, as well as Bjørn-Helge Mevik for fruitful discussion.

Bibliography

J. M. Abowd, F. Kramarz, and D. N. Margolis. High wage workers and high wage firms. *Econometrica*, 67(2):251–333, 1999. doi: 10.1111/1468-0262.00020. URL http://www.jstor.org/stable/2999586. [p105, 108]

J. M. Abowd, R. H. Creecy, and F. Kramarz. Computing person and firm effects using linked longitudinal employer-employee data. Longitudinal Employer-Household Dynamics Technical Papers 2002-06, Center for Economic Studies, U.S. Census Bureau, 2002. URL http://ideas.repec.org/p/cen/tpaper/2002-06.html. [p108]

L. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68:337–404, 1950. URL http://www.ams.org/journals/tran/1950-068-03/S0002-9947-1950-0051437-7. [p112]

C. Badea, S. Grivaux, and V. Müller. The rate of convergence in the method of alternating projections. *St. Petersburg Mathematical Journal*, 23:413–434, 2012. URL http://www.ams.org/journals/spmj/2012-23-03/S1061-0022-2012-01202-1. preprint in arXiv:1006.2047. [p112]

D. Bates and M. Maechler. *Matrix: Sparse and dense matrix classes and methods*, 2013. URL http://CRAN.R-project.org/package=Matrix. R package version 1.0-15. [p105]

H. H. Bauschke, F. Deutsch, H. Hundal, and S.-H. Park. Accelerating the convergence of the method of alternating projections. *Transactions of the American Mathematical Society*, 355(9):3433–3461, 2003. URL http://www.ams.org/journals/tran/2003-355-09/S0002-9947-03-03136-2. [p112, 115]

A. Carneiro, P. Guimarães, and P. Portugal. Real wages and the business cycle: Accounting for worker and firm heterogeneity. *American Economic Journal: Macroeconomics*, 4(2):133–152, Apr. 2012. doi: doi:10.1257/mac.4.2.133. URL http://www.ingentaconnect.com/content/aea/aejma/2012/00000004/00000002/art00005. [p108]

T. Cornelißen. The Stata command felsdvreg to fit a linear model with two high-dimensional fixed effects. *Stata Journal*, 8(2):170–189, 2008. URL http://www.stata-journal.com/article.html?article=st0143. [p115]

Y. Croissant and G. Millo. Panel data econometrics in R: The plm package. *Journal of Statistical Software*, 27(2), 2008. URL http://www.jstatsoft.org/v27/i02/. [p105]

G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006. URL http://igraph.sf.net. [p114]

F. Deutsch and H. Hundal. The rate of convergence for the method of alternating projections, II. *Journal of Mathematical Analysis and Applications*, 205(2):381–405, 1997. ISSN 0022-247X. doi: 10.1006/jmaa.1997.5202. URL http://www.sciencedirect.com/science/article/pii/S0022247X97952021. [p112, 115]

J. A. Eccleston and A. Hedayat. On the theory of connected designs: Characterization and optimality. *The Annals of Statistics*, 2(6):1238–1255, 1974. URL http://www.jstor.org/stable/2958341. [p108]

S. Gaure. OLS with multiple high dimensional category variables. *Computational Statistics & Data Analysis*, 66:8–18, 2013a. ISSN 0167-9473. doi: http://dx.doi.org/10.1016/j.csda.2013.03.024. URL http://www.sciencedirect.com/science/article/pii/S0167947313001266. [p106, 107, 108, 114, 115]

S. Gaure. *lfe: Linear group fixed effects*, 2013b. URL http://CRAN.R-project.org/package=lfe. R package version 1.5-1107. [p105]

W. B. Gearhart and M. Koshy. Acceleration schemes for the method of alternating projections. *Journal of Computational and Applied Mathematics*, 26(3):235–249, 1989. ISSN 0377-0427. doi: 10.1016/0377-0427(89)90296-3. URL http://www.sciencedirect.com/science/article/pii/0377042789902963. [p112]

J. D. Godolphin and E. J. Godolphin. On the connectivity of row-column designs. *Utilitas Mathematica*, 60:51–65, 2001. [p109]

P. Guimarães. REG2HDFE: Stata module to estimate a linear regression model with two high dimensional fixed effects. Statistical Software Components, Boston College Department of Economics, 2009. URL http://ideas.repec.org/c/boc/bocode/s457101.html. [p115]

P. Guimarães and P. Portugal. A simple feasible procedure to fit models with high-dimensional fixed effects. *Stata Journal*, 10(4):628–649, 2010. URL http://www.stata-journal.com/article.html?article=st0212. [p106, 115]

I. Halperin. The product of projection operators. *Acta Scientiarum Mathematicarum (Szeged)*, 23(1-2):96–99, 1962. URL http://acta.fyx.hu/acta/showCustomerArticle.action?id=7164&dataObjectType=article. [p106]

N. J. Higham. Analysis of the Cholesky decomposition of a semi-definite matrix. In M. G. Cox and S. J. Hammarling, editors, *Reliable Numerical Computation*, chapter 9, pages 161–185. Oxford University Press, 1990. eprint: http://eprints.ma.man.ac.uk/1193. [p111]

A. Kaczmarz. Angenäherte Auflösung von Systemen linearer Gleichungen. *Bulletin International de l'Academie Polonaise des Sciences et des Lettres. Classe de Sciences Mathématiques et Naturelles.*, 35: 355–357, 1937. Série A, Sciences Mathématiques. [p107]

S. Kayalar and H. L. Weinert. Error bounds for the method of alternating projections. *Mathematics of Control, Signals and Systems*, 1:43–59, 1988. doi: 10.1007/BF02551235. URL http://link.springer.com/article/10.1007%2FBF02551235. [p112]

S. Markussen. Personal Communication, 2013. [p115]

S. Markussen and K. Røed. Social insurance networks. IZA Discussion Papers 6446, Institute for the Study of Labor (IZA), 2012. URL http://ideas.repec.org/p/iza/izadps/dp6446.html. [p105, 115]

A. Ouazad. A2REG: Stata module to estimate models with two fixed effects. Statistical Software Components, Boston College Department of Economics, 2008. URL http://econpapers.repec.org/RePEc:boc:bocode:s456942. [p115]

StataCorp. *Stata Data Analysis Statistical Software: Release 13*. StataCorp LP, College Station, TX, 2013. URL http://www.stata.com/. [p115]

S. M. Torres, P. Portugal, J. T. Addison, and P. Guimarães. The sources of wage variation: A three-way high-dimensional fixed effects regression model. IZA Discussion Paper 7276, Institute for the Study of Labor (IZA), 2013. URL http://ssrn.com/abstract=2238309. [p105, 108]

S. Urbanek. *multicore: Parallel processing of R code on machines with multiple cores or CPUs*, 2011. URL http://CRAN.R-project.org/package=multicore. R package version 0.1-7. [p115]

D. L. Weeks and D. R. Williams. A note on the determination of connectedness in an N-way cross classification. *Technometrics*, 6(3):319–324, 1964. doi: 10.1080/00401706.1964.10490188. URL http://www.tandfonline.com/doi/abs/10.1080/00401706.1964.10490188. [p108, 109]

J. M. Wooldridge. *Econometric Analysis of Cross Section and Panel Data*. The MIT Press, 2002. [p105, 111]

Simen Gaure
Ragnar Frisch Centre for Economic Research
Oslo, Norway
Simen.Gaure@frisch.uio.no

The R in Robotics

rosR: A New Language Extension for the Robot Operating System

by André Dietrich, Sebastian Zug, and Jörg Kaiser

Abstract The aim of this contribution is to connect two previously separated worlds: robotic application development with the Robot Operating System (ROS) and statistical programming with R. This fruitful combination becomes apparent especially in the analysis and visualization of sensory data. We therefore introduce a new language extension for ROS that allows to implement nodes in pure R. All relevant aspects are described in a step-by-step development of a common sensor data transformation node. This includes the reception of raw sensory data via the ROS network, message interpretation, bag-file analysis, transformation and visualization, as well as the transmission of newly generated messages back into the ROS network.

Introduction

The development of robotic applications has become more and more an interdisciplinary task, ranging from mechanical and electrical engineering, signal processing, control technology and cybernetics, up to artificial intelligence, etc. The Robot Operating System[1], abbreviated as ROS, is a relatively new framework, intended to simplify the development of any kind of robotic application by bringing together the latest scientific knowledge. And by the time it has become a de facto standard in the scientific robotics community (cf. Kramer and Scheutz, 2007). In order to disseminate a certain algorithm, toolbox, or function, a large number of developers provide interfaces to ROS. Currently there are more than 1700 different packages available online for various purposes, for example "Marker Detection", "Simultaneous Localization And Mapping", "Trajectory Planning", and many more. These packages can easily be used, combined, and integrated into new applications, with only a little knowledge about the ROS philosophy. Robotic applications are no longer designed as single and monolithic processes, but instead as a collection of nodes, which perform a certain type of computation, similar to the UNIX philosophy (cf. Salus, 1994): "*Write programs that do one thing and do it well. Write programs to work together. Write programs to ...*". The development of a ROS node is mainly based on two philosophical pillars (apart from open source ideology), namely multilingualism and peer-to-peer (cf. Quigley et al., 2009).

Multilingualism means to implement a node in the most appropriate programming language. In most cases C++ (roscpp[2]) is the ideal choice for developing a node (cf. Quigley et al., 2013), especially to fulfill hardware-related or computationally expensive functionalities, whereas Python (rospy[2]) is suitable for rapid prototyping, scripting, and GUI development. The Lisp language support (roslisp[2]) was also intended for scripting, but is mostly employed for fast and interactive debugging. Next to these programming languages with full ROS support, there are also other "experimental" language[3]-extensions, serving different needs. Java (rosjava[2]) allows developers to build nodes for Android, while Lua (roslua[2]) currently is used for scripting reactive robot behavior.

The interconnection between nodes is established at run time in a peer-to-peer topology, using publish/ subscribe or service-based communication. This allows to run nodes on different hosts and it enables their dynamic integration and segregation. For this purpose, ROS guarantees mutual understanding (and thus interoperability) by explicit message descriptions.

Until now, it was not possible to develop nodes in the statistical programming language R. R is probably not the best choice for implementing fancy GUIs or to develop hardware drivers, but it has by far one of the richest libraries for statistical analyses, data-mining, machine learning, and visualization. According to the comparison of statistical packages on Wikipedia (2013), R is the only product that has support for every listed demand (unlike MATLAB or SciPy). Furthermore, most of its algorithms are directly implemented in C or C++, which guarantees fast and efficient computation.

In light of these qualities, R is an adequate programming language for sensor signal analysis, evaluation and visualization. These types of tasks are very important for embedded or robotic applications where developers cope with non-linear behavior, different measurement faults or external disturbances. An exemplary evaluation of sensor data is described in Dietrich et al. (2010). The presented infrared sensors can be easily disturbed by external light, leading to faulty measurements. But these external disturbances also affect the measurement noise in such a way that it can be detected by applying statistical tests – an ideal application for R.

[1]Official ROS project website: `http://www.ROS.org`
[2]ROS language extensions for C++, Python, Lisp, Java, and Lua.
[3]For a complete list of supported programming languages see `http://www.ros.org/wiki/ClientLibraries`.

Overview

This work is intended to serve two tasks. On the one hand, we want to introduce ROS and present its general development principles to the common R developer. On the other hand, we want to convince ROS developers to use R for certain types of computation. To demonstrate the utility of combining ROS and R, we chose a quite common application, the linearization of sensor distance measurements. In a tutorial-like section we will highlight all relevant aspects in a step-by-step development of an R node. This node will receive raw sensor data, perform a linearization, visualize and publish the resulting values.

Before we describe how to develop a ROS node in pure R and how to install our **rosR** package, we will give a brief introduction to the common ROS communication principles in the next section. Some limitations of R made it difficult to implement **rosR** as a typical R library, such as single threading, problems in supporting sockets and in handling raw data streams. Instead we developed a common ROS package, which furthermore required to interface several ROS packages. To ease the interconnection of ROS C++ functionality with R we applied the Simplified Wrapper and Interface Generator (SWIG-2.0, 2013). The second-last section deals with these implementation details as well as the project structuring and should afterwards allow to extend the package with custom functionality. We conclude our paper with a summary.

ROS communication principles

As already mentioned, ROS offers two basic communication principles, publish/subscribe and service-based communication. The participation of an R node within a ROS network requires the implementation of these paradigms. Hence, we briefly introduce both concepts, although **rosR** currently does not support services.

Publish/Subscribe

The main idea behind publish/subscribe (pub/sub) is to decouple the production and consumption of data. In contrast to the widely used address-based communication paradigm, it offers content-based communication. But, as in most cases including ROS pub/sub, this is just an overlay on address-based communication. Topics in this sense define logical channels that transport all related messages. In ROS, a channel is tagged by a unique string, such as '`/map`', '`/odometry`', or '`/laserscan`', which identifies the content of a channel and its message format. Messages are strongly typed data structures, composed of different primitive types (e. g., `string`, `integer`, `float`, `array`, etc.), similar to structs in C. ROS provides a number of standard data types for certain sensors or tasks. But a user can define individual or adapted message formats related to specific purposes. A publisher (producing node) advertises its intent to publish on a topic, while a subscriber (consuming node) indicates its interest on one or more topics. In ROS there is a central master node that provides name registration and lookup for all connected nodes. The steps for establishing a connection been both parties are depicted in Figure 1. Thus, data is only transmitted, if there is at least one publisher and one subscriber for one topic.

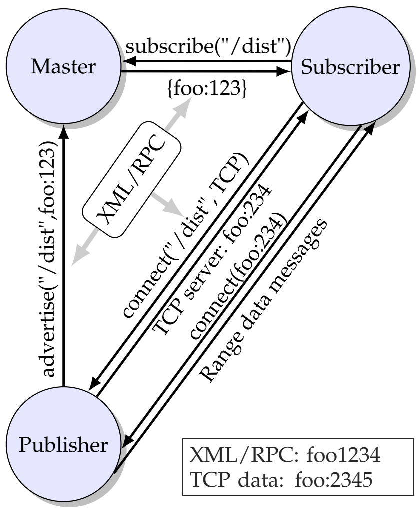


Figure 1: Communication initialization between a publishing and a subscribing node.

Services

Pub/sub is ideal for dynamic and flexible message passing, but in some cases it might be more useful to request for a certain type of data (e. g., camera parameters, robot states, etc.) or the execution of an action (e. g., grasping an object, planning a trajectory, etc.). This kind of remote procedure call is handled via services. Nodes offer their services by using string names, similar to topics in pub/sub, and clients can call these services by sending their request in an appropriate message format, awaiting the reply message. The connection between both entities is also instantiated with the help of the master, as seen in Figure 1.

Parameters

The access to the parameter server is often mentioned as a third communication method, comparable to shared memory. Parameters are stored at runtime in a multivariate dictionary, which is also hosted by the ROS master node. It is mainly applied for simple configuration purposes, but it also allows to inspect and to modify the global configuration.

Installation

First of all ROS has to be installed and configured. Good installation guides can be found on `http://www.ros.org/wiki/ROS/Installation`. Additionally, we put an extensive installation description for Ubuntu 12.04 and ROS "Groovy Galapagos" on our project site `http://www.ros.org/wiki/rosR#R-Side`. The install manual was intended to guide R developers with no or only a little ROS experience. Information on further developments and adaptations for new ROS versions will be available on the project website too. Before compiling our **rosR** package, there are three things required (apart from an ROS-base and a R-base):

1. SWIG is responsible for interfacing the C++ code (only SWIG2.0 is able to generate appropriate wrappers for R),
2. the R development packages for C++ **Rcpp** (cf. Eddelbuettel and François, 2011),
3. and a running subversion client to be able to download our package via:
   `$ svn co http://svn.code.sf.net/p/ivs-ros-pkg/code/trunk/rosR`

Finally, simply enter the installation folder of **rosR** and run the shell-command `rosmake` (cf. Foote, 2013). If everything compiled successfully, you should be able to launch a simple test application with:

```
$ roslaunch rosR random.launch
```

How-to develop a rosR node

As already mentioned, this section is intended to demonstrate how to use the **rosR** API, by developing a node responsible for a common sensor data transformation. For this purpose we chose a common infrared distance sensor, the Sharp GP2D120 (cf. Sharp Cooperation, 2007). We start with a very basic application and extend it in the following subsections. The complete source code for every example as well as the attached sensor measurements can be downloaded from the following ROS project repository (which can be treated as any ordinary ROS package and therefore has to be built with rosmake:

```
$ svn co http://svn.code.sf.net/p/ivs-ros-pkg/code/trunk/rosR_demos
```

Every subsection will tackle an individual development part and in doing so, will also explain some internal ROS matters, which makes it necessary to switch between the R-terminal and the command-line shell. '$' is used to indicate the shell usage '>' and '+' indicate R-commands, while '#' is used for comments in both cases.

With the following shell-command you start the replay of previously recorded ROS data, gathered from a Sharp infrared distance sensor. It will also start a ROS master. Therefore, the user should not cancel it, because then it will not be possible to subscribe to any kind of data or to publish any sensor data.

```
$ # replay of data (using rosbag) from rosR_demos/nodes/RJournal/sharpGP2D120.bag
$ roslaunch rosR_demos sharp-playbag.launch
```

Node initialization

Because we have developed a typical ROS package, which can be located anywhere on your system, it has to be loaded manually. Run the following command in R to load all required functionality. It invokes the system call '`rospack find`' (cf. Gerkey et al., 2013), which is responsible for locating the current **rosR** installation directory.

```
> source(paste(system("rospack find rosR", intern = TRUE), "/lib/ros.R", sep = ""),
+   chdir = TRUE)
```

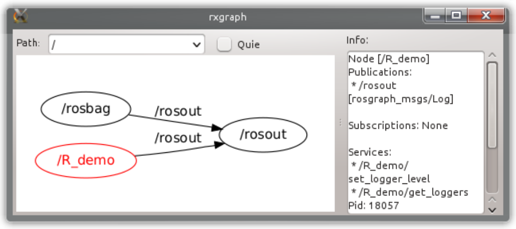
Each command of our API starts with 'ros.', so that the auto-completion, after entering 'ros.' into the R-terminal, should present the same result. These are all functions that are required to participate in every ROS network and to publish and subscribe for messages.

```
> ros.
ros.BagRead      ros.Logging          ros.ParamType                     ros.TimeNow
ros.Debug        ros.Message          ros.Publisher                     ros.Warn
ros.Error        ros.OK               ros.ReadMessage                   ros.WriteMessage
ros.Fatal        ros.ParamDelete      ros.SpinOnce
ros.Info         ros.ParamGet         ros.Subscriber
ros.Init         ros.ParamSet         ros.SubscriberHasNewMessage
```

First of all, and as it is required for all ROS nodes, the node has to be registered as a new participant by announcing at the ROS master with an arbitrary but unique identifier. This can be done with the R-command:

```
> ros.Init("R_demo")
```

ROS also provides a monitoring tool rxgraph (cf. Conley, 2013b) that shows all active applications ("nodes") and active connections between them. Starting this tool from the shell should show the same result, as presented in Figure 2a below. By examining these nodes, you will see that our newly created node is neither subscribed to nor publishing on a certain topic, while rosbag (cf. Field et al., 2013) is continuously publishing Float32 data under topic '/sharpGP2D120/Voltage'.

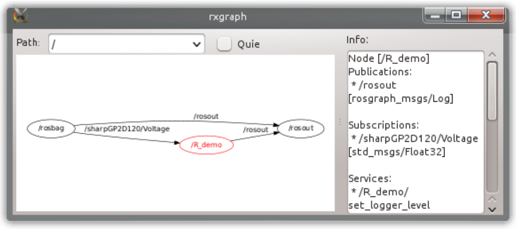


```
Node [/R_demo]
Publications:
 * /rosout
[rosgraph_msgs/Log]

Subscriptions: None

Services:
 * /R_demo/
set_logger_level
 * /R_demo/get_loggers
Pid: 18057
```

(a) before subscribing



```
Node [/R_demo]
Publications:
 * /rosout
[rosgraph_msgs/Log]

Subscriptions:
 * /sharpGP2D120/Voltage
[std_msgs/Float32]

Services:
 * /R_demo/
set_logger_level
```

(b) after subscribing

Figure 2: Connectivity graphs, created with the shell-command rxgraph, both show additional information about node "R_demo" on the right.

Subscriptions and logging

Within this section, we will show how to subscribe to and receive messages from other ROS nodes. A subscription in R can be created as easily as in other programming languages. The only relevant parameters are the topic and the message type (we will handle messages in more detail within subsection "Publishing new messages"). The function call as listed below creates a new subscription and also changes the connectivity graph of the ROS network, as depicted in Figure 2b.

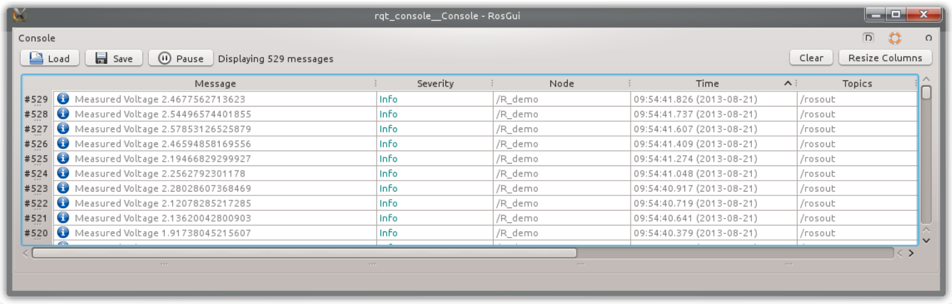
```
> subscription <- ros.Subscriber("/sharpGP2D120/Voltage", "std_msgs/Float32")
```

Due to the fact that R is single-threaded and the lack of native support for callbacks, we have to continuously poll for new messages. The code snippet below faciliates this:

```
> while(ros.OK()) {  # evaluates to TRUE as long as the master is online
+     ros.SpinOnce() # fill the subscription buffers with new messages
+     if(ros.SubscriberHasNewMessage(subscription)) {
+         message <- ros.ReadMessage(subscription)
+         ros.Info( paste("Measured Voltage", message$data) )
+ } }
[ INFO] [1374149363.939419670]: Measured Voltage 0.675665080547333
[ INFO] [1374149364.069367143]: Measured Voltage 0.713892936706543
...
```

Function ros.SpinOnce() is responsible for filling the message buffers of all generated subscribers with the newest message at once. The fact that a subscriber has received a new message or not, is indicated with the function call ros.SubscriberHasNewMessage(). This has to be called for every subscription. The last message that was received can be read out with function ros.ReadMessage(). A message remains in the message buffer of a subscriber as long as it is not overwritten by a newly received message (ros.SpinOnce()).

Instead of using the standard `print` function of R to print out the content of the message, it is also possible to use some of ROS's logging functionalities `ros.Logging(text,mode)` or the abbreviations for different modes. These are `ros.Debug(text)`, `ros.Info(text)`, `ros.Warn(text)`, `ros.Error(text)`, and `ros.Fatal(text)`. The usage of these functions enable a user to debug an application, consisting of multiple nodes, since this information is published to node '`rosout`' and can be further analyzed with various tools, compare with Figure 3.


rqt_console__Console - RosGui

Console

Load | Save | Pause | Displaying 529 messages | Clear | Resize Columns

	Message	Severity	Node	Time	Topics
#529	Measured Voltage 2.4677562713623	Info	/R_demo	09:54:41.826 (2013-08-21)	/rosout
#528	Measured Voltage 2.54496574401855	Info	/R_demo	09:54:41.737 (2013-08-21)	/rosout
#527	Measured Voltage 2.57853126525879	Info	/R_demo	09:54:41.607 (2013-08-21)	/rosout
#526	Measured Voltage 2.46594858169556	Info	/R_demo	09:54:41.409 (2013-08-21)	/rosout
#525	Measured Voltage 2.19466829299927	Info	/R_demo	09:54:41.274 (2013-08-21)	/rosout
#524	Measured Voltage 2.2562792301178	Info	/R_demo	09:54:41.048 (2013-08-21)	/rosout
#523	Measured Voltage 2.28028607368469	Info	/R_demo	09:54:40.917 (2013-08-21)	/rosout
#522	Measured Voltage 2.12078285217285	Info	/R_demo	09:54:40.719 (2013-08-21)	/rosout
#521	Measured Voltage 2.13620042800903	Info	/R_demo	09:54:40.641 (2013-08-21)	/rosout
#520	Measured Voltage 1.91738045215607	Info	/R_demo	09:54:40.379 (2013-08-21)	/rosout

Figure 3: Screenshot of `rqt_console` (cf. Blasdel, 2013), a viewer that displays logs of various nodes.

Executable scripts

Two things are required, to start the example from above as a R-script from the shell. A comment at the first line of the script, which defines the correct interpreter:

```
#!/usr/bin/r
```

The script also has to be made executable, which can be done with the shell-command '`chmod`':

```
$ chmod +x script.R
```

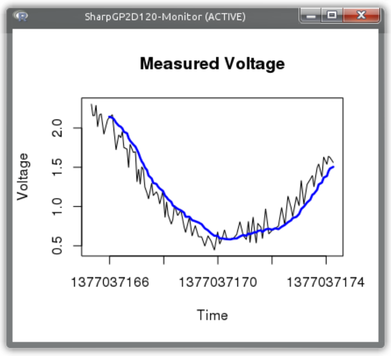
Afterwards it is possible to run the script in the same way as any other ordinary ROS node. The script that we had developed so far can be found at 'rosR_demos/nodes/RJournal/demo0.R' (this folder also contains further examples) and is started from the shell with `rosrun` (cf. Leibs and Kruse, 2013):

```
$ rosrun rosR_demos demo0.R
```

Time and visualization

Since we started out by stating how easy it is to visualize data with R, the snippet below shows an improved version of the previous code. The voltage values and their reception time are stored within two vectors, which are updated and plotted during every processing cycle. Furthermore, a filter is applied to smooth the results of the measurements. The resulting visualization is shown in Figure 4 and the corresponding source code can be examined at 'rosR_demos/nodes/RJournal/demo1.R'.

```
> x11(width = 6, height = 4,
+     title = "SharpGP2D120-Monitor")
> Voltage <- Time <- rep(NA, 100)
> while(ros.OK()) {
+   ros.SpinOnce()
+   if (ros.SubscriberHasNewMessage(subscription)){
+     message <- ros.ReadMessage(subscription)
+     Voltage <- c(Voltage[-1], message$data)
+     Time <- c(Time[-1], ros.TimeNow())
+     plot(Time, Voltage, t = "l",
+          main = "Measured Voltage")
+     lines(Time,
+       filter(Voltage, rep(0.1, 10), sides = 1),
+              type = "l", col = "blue", lwd = 2.5)
+ }}
```



F. 4: `$ rosrun rosR_demos demo1.R`

The only newly used ROS function is `ros.TimeNow()`. It returns the global system time of the ROS master as `double` value, as presented below. The dot separates between seconds since the first of January 1970 and nanoseconds (everything after the decimal point).

```
> as.POSIXlt(ros.TimeNow(), origin="1970-01-01")
[1] "2013-07-18 15:46:26 CEST"
```

```
> ros.TimeNow()
[1] 1374155184.932926893234
```

Bag-files and linearization

The conversion of non-linear voltage outputs into usable distance measurements is a frequently occurring tasks, by dealing with raw sensor data. The data sheet from the Sharp Cooperation (2007) provides a voltage-distance characteristic, which is quite similar to our measurements depicted in Figure 5. These measurements (sensor output voltage and manually controlled distance) were stored additionally in the bag-file that is currently replayed by rosbag to publish the voltage values of Sharp distance sensor. A bag is a standard format in ROS for storing any kind of ROS messages. Reading in messages from a bag-file, to analyze their content in R, is therefore also provided by our API. Next to replaying messages, rosbag can also be used for recording messages and inspecting bag-files, as it is presented below:

```
$ rosbag info rosR_demos/nodes/RJournal/sharpGP2D120.bag
...
start:       Nov 19 2013 14:03:28.91 (1384866208.91)
end:         Nov 19 2013 14:28:42.09 (1384867722.09)
size:        39.6 KB
messages:    554
compression: none [1/1 chunks]
types:       rosR_demos/Linearization [3f7dd391cdbb9d1f72822c152e8c430f]
             std_msgs/Float32         [73fcbf46b49191e672908e50842a83d4]
topics:      /sharpGP2D120/Linearization   240 msgs    : rosR_demos/Linearization
             /sharpGP2D120/Voltage         314 msgs    : std_msgs/Float32
```

The result shows that there are two different topics in two different message formats. The voltage values to which we had subscribed '/sharpGP2D120/Voltage' and '/sharpGP2D120/Linearization', which contains the previously measured distance/voltage values. This message format is defined in 'rosR_demos/msg/Linearization.msg' and is composed of two Float32 values. Reading these messages from the bag-file into R can be done with the function ros.BagRead. The required input parameters are the filename and a vector of strings that defines the topics of interest:

```
> file <- paste(system("rospack find rosR_demos", intern = TRUE),
+                "/nodes/RJournal/sharpGP2D120.bag", sep = "")
> bag.data <-  ros.BagRead(file, c("rosR_demos/Linearization"))
```

The result is a list consisting of four vectors, these are the topic names, message types, timestamps, and the messages themselves (the details of message conversion are explained in the next subsection):

```
> bag.data$topic[2]                           > bag.data$message[2]
[1] "/sharpGP2D120/Linearization"             [[1]]
> bag.data$data_type[2]                       [[1]]$dist
[1] "rosR_demos/Linearization"                [1] 0.031
> bag.data$time_stamp[2]                      [[1]]$volt
[1] 1374499648.961932659149                   [1] 3.0251
```

Before fitting a suitable equation to describe the relation between distance and voltage, we have to transform the linearization messages into an appropriate R format (data.frame). In the code example below, this happens by extracting all required values with the help of sapply:

```
> # Copy data
> dist <- sapply(bag.data$message,
+                "[[", "dist")
> volt <- sapply(bag.data$message,
+                "[[", "volt")
> sharp.data <- data.frame(dist, volt)
> plot(sharp.data, main = "Linearization")
>
> # Linearization
> sharp.reg <- lm(dist ~ poly(volt, 8),
+                 data = sharp.data)
> sharp.dist <- function(volt) {
+  predict(sharp.reg,data.frame(volt))
+ }
>
> # Test 1              > # Test 2
> sharp.dist(0.8)       > sharp.dist(2.0)
[1] 0.1638513           [1] 0.0613970
```

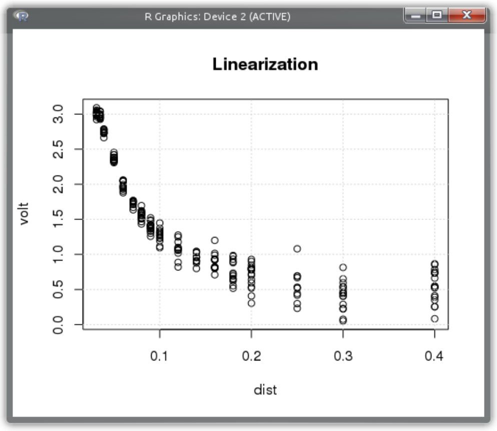


Figure 5: > plot(sharp.data,main='Lin...')

We decide to approximate the relation by a polynom. Previous investigations showed, that a polynom with a degree of 8 meets our expectations related to the maximum deviation. This is pretty straight forward in R and can be realized with `lm` within one line. The following line defines a function that makes predictions based on the fitted polynom. The predictions made by the resulting linearization function `sharp.dist` are quite close to the measurements in data sheet Sharp Cooperation (2007). The confidence intervals of this linearization are depicted in Figure 6. See also 'rosR_demos/nodes/RJournal/demo2.R', which contains the source code of the running example.

Publishing new messages

This subsection concludes our attempts at developing ROS nodes in R. While the first part describes message handling and conversion from ROS to R, the second part will show how simply the calculated distance values can be published under a new topic and in an appropriate message format ('`sensor_msgs/Range`').

Messages

In ROS every topic is defined by a strongly typed message format. Messages are therefore described in language-neutral interface definition format. They are defined as compositions of primitive data types such as `int32`, `float64`, `string`, `arrays`, etc., but they can also contain other message definitions to derive more complex structures for various purposes. Furthermore, every ROS package can define its own message formats. Addressing a specific message definition therefore requires two values, in the format of `"ros-package-name/message-description-file"`. By applying `rosmsg` (cf. Conley and Foote, 2013) it is possible to examine messages that are defined in different packages as follows:

```
$ rosmsg show rosR_demos/Linearization
float32 dist
float32 volt

$ rosmsg show sensor_msgs/Range
uint8 ULTRASOUND=0
uint8 INFRARED=1
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
uint8 radiation_type
float32 field_of_view
float32 min_range
float32 max_range
float32 range
```

```
$ rosmsg show std_msgs/Float32
float32 data

$ rosmsg show sensor_msgs/LaserScan
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
float32 range_max
float32[] ranges
```

The listing above shows the definition formats of four different messages. The first two of them were already used and you will probably recognize where the names for list elements ('`$data`', '`$dist`', '`$volt`') came from. It is notable that the more complex messages combine static information related to the sensor ('`field_of_view`', '`min_range`') and the actual measurement sets ('`stamp`', '`range`'). All ROS messages are automatically converted into compositions of lists, whose format is defined by the message definition. The resulting list from reading in a message, from a subscription or from a bag-file, is already presented in the appropriate structure. Calling function `ros.Message` will also generate an empty message in the requested format:

```
> range <- ros.Message("sensor_msgs/Range")
```

Elements of that newly generated message can simply be changed by assigning new values as follows (range values from Sharp Cooperation (2007)):

```
> # equal to range.min_range as in rospy
> range$min_range <- 0.04
> range$max_range <- 0.30
```

```
> # or to range->min_range in roscpp
> range$header$seq <- 0
> range$header$frame_id <- "/sharp"
```

'`header`' in this case is an example of a nested message definition. As shown in the result of rosmsg, this element of type '`std_msgs/Header`' contains three primitive data types, which means in R that '`header`' is also translated into a list containing three further elements. Some message definitions like '`sensor_msgs/LaserScan`' might also contain arrays, which cannot be directly translated into appropriate R elements (vectors), as it happens for primitive data types. The conversion of arrays by using SWIG2.0 can be very time consuming, at least in R. It requires that every single element has to be copied into a new R vector. Just think of a camera frame with 640x480 (RGB) pixels, resulting in an array with 921600 elements. We therefore choose another strategy, to enable and to speed up the access to these values, if it is required. An array is handled in background as a C++ `std::vector<T>`. The access to the elements of this C++ vector is enabled via R wrapper classes. See file 'rosR/lib/std_vector.R' for implementation details.

```
> scan <- ros.Message("sensor_msgs/LaserScan")
> typeof(scan$ranges) # element defines float32 array
[1] "S4"
```

The functions that are currently used to wrap the access to these C++ vectors are `length`, `print`, `show`, `resize`, `pop`, `pop_back`, `back`, `front`, `clear`, and `[]`. The usage of these functions allows to query the length or to print the content of an array as follows:

```
> length(scan$ranges)                        > scan$ranges
[1] 0                                        [1] NULL
```

Inherited from the SWIG2.0 conversion of `std::vectors`, array elements have to be added by using the function append or push_back and can be removed with the functions pop or pop_back, while accessing and changing already existing array elements can be done in the common R way:

```
> append(scan$ranges, 0)                     > msg$ranges[2:4]
> append(scan$ranges, c(1, 2, 3))            [1] 1 2 3
> length(scan$ranges)                        > msg$ranges[2:4] <- c(11, 22, 33)
[1] 4                                        > msg$ranges[1:5]
> push_back(scan$ranges, c(4, 5, 6))         [1] 0 11 22 33 4
> length(scan$ranges)                        > pop(msg$ranges)
[1] 7                                        [1] 6
> msg$ranges                                 > msg$ranges
[1] 0 1 2 3 4 5 6                            [1] 0 11 22 33 4 5
```

It has to be noted that `[]` always creates local copies of the elements stored within a C++ vector. Thus, accessing and analyzing huge arrays (e. g., camera images) is possible but it may require some time for conversion.

```
> sum(scan$ranges)
Error in sum(scan$ranges) : invalid 'type' (S4) of argument
> sum(scan$ranges[1:6])
[1] 75
```

Nevertheless, it is possible to speed up the execution of required functions, such as `sum`, `median`, or `density`, by manually implementing wrapper functions in the same way as done for `length` or `[]` in file 'rosR/lib/std_vector.R'. See therefore also section "Implementation details".

Publishing

Coming back to our example, the publication of converted distance measurements requires the announcement of a new topic with the new message format. This is done automatically by creating a new publication:

```
> publication <- ros.Publisher("/sharpGP2D120/Distance", "sensor_msgs/Range")
```

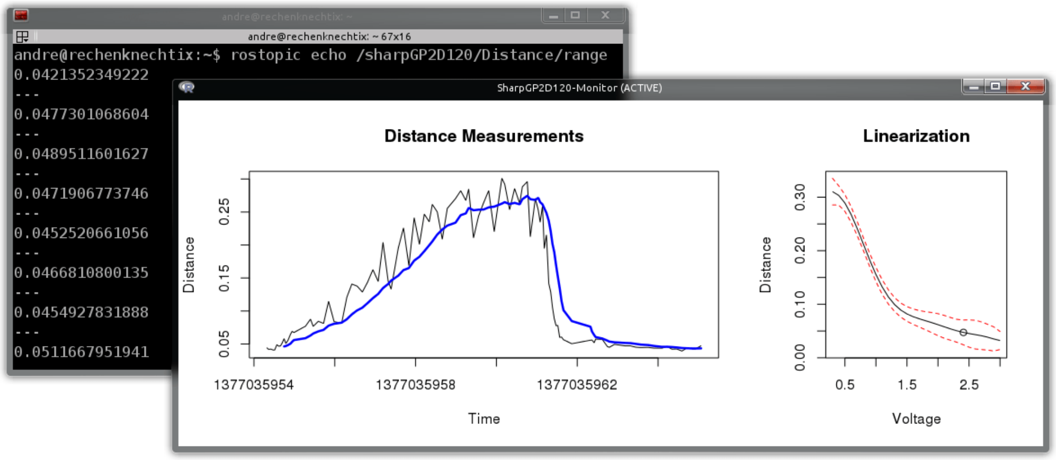
Afterwards, the previously generated message of type '`sensor_msgs/Range`' has to be updated with the current distance value, a timestamp, and a sequence number. This can happen within the main loop as follows:

```
> range$range <- sharp.dist(volt)            # convert voltage to distance
> range$header$seq <- counter                # store a running sequence number
> range$header$stamp <- ros.TimeNow()        # store the current time
```

Finally, the message can be published by writing the message to the publisher. In contrast to a subscription, this information is immediately published and does not require to run function `ros.SpinOnce()`.

```
> ros.WriteMessage(publication, range)       # publish the adapted message
```

Figure 6 shows the running example of the R node, which is publishing and visualizing the linearized distance measurements. The additional plot on the left shows the non-linear relation between voltage and distance as well as the current measurement. See also 'rosR_demos/nodes/RJournal/demo3.R'.



```
andre@rechenknechtix:~$ rostopic echo /sharpGP2D120/Distance/range
0.0421352349222
---
0.0477301068604
---
0.0489511601627
---
0.0471906773746
---
0.0452520661056
---
0.0466810800135
---
0.0454927831888
---
0.0511667951941
```

Figure 6: Screenshots of the final R application (foreground: `$ rosrun rosR_demos demo3.R`) and the published range messages in a textual format (background: `$ rostopic echo /sharp-GP2D120/Distance`) by using `rostopic` (cf. Conley, 2013a).

Accessing the parameter server

The parameter server is also part of the ROS master and allows different nodes to exchange simple parameters (i. e., `logical`, `integer`, `double`, `character`). Accessing and storing parameters is enabled by the functions `ros.ParamSet`, `ros.ParamGet`, `ros.ParamType`, and `ros.ParamDelete`, which can be used as follows:

```
> ros.ParamSet("name", "value")
> ros.ParamGet("name")
[1] "value"
> ros.ParamSet("name", TRUE)
> ros.ParamType("name")
[1] "logical"
> ros.ParamSet("name", 12.22)
```

```
> ros.ParamGet("name")
[1] 12.22
> ros.ParamType("name")
[1] "double"
> ros.ParamDelete("name")
> ros.ParamType("name")
NULL
```

Function `ros.ParamType` checks the type of a certain parameter, but can also be used to check if a parameter does not exist, in this case it will return `NULL`.

Implementation details

This section will give insights into the project structure and details of implementation. Thus, adapting and adding new functionality is strongly recommended. As mentioned earlier, our package was developed with the help of SWIG2.0. This is an "easy to use" tool for interfacing C and C++ libraries/programs from various languages such as Python, Tcl, Go, Guile, Java, and also R. Literally, it can be seen as a compiler that takes C and C++ declarations as input and generates wrappers as output. These interfaces and the conversion between different data formats for R are automatically generated with the help of the **Rcpp** package, which provides required R functions and a C++ library for the interconnection.

Structure

To ease the development and the maintenance of the code and the API, the **rosR** project was separated into an R and a C++ part:

- 'rosR/lib/': Folder for all R related files that are listed below.
  - 'ros.R': Is the main file of the project, which has to be loaded by every R node. It implements the whole **rosR** API and is therefore also responsible for loading other R files. It also contains a couple of internal helper functions, responsible for message conversion and handling. Extending the **rosR** API therefore also requires adapting this file.
  - 'rosR.R' and 'rosR.so': This is the SWIG2.0 generated wrapper code for R and the compiled C++ library, which enables to access ROS via C++ objects and methods.

    - 'std_vector.R': This is the handmade wrapper for accessing ROS arrays. It implements a general 'rros_vector' class as well as classes for every primitive data type (std::vector<T>). Based on these class definitions, different wrapper functions were implemented (i. e., length, print, show, resize, pop, pop_back, back, front, clear, and []) to ease and hide the complexity of array access. This is also the right place to include additional user-defined functionality such as sum, mean, etc., which can increase the speed of execution. Vector elements can be accessed and manipulated directly with C++ functions, which is probably more advantageous than creating local vector copies.

- 'rosR/src/': Includes all C++ wrappers as listed below, based on roscpp by Quigley et al. (2013), topic_tools by Quigley and Gerkey (2013), and rosbag by Field et al. (2013).
    - 'rosR.i': The SWIG2.0 input file, used to include all below listed classes. The result is the generated wrapper file 'rosR_wrap.cpp', which allows to access all methods and classes, defined in the files below.
    - 'rosR.h/cpp': Defines general functionality such as initializing a node and generating a node handle, logging, timing, and spinning. It is the right place for including basic functionality.
    - 'PublisheR.h/cpp': The generation of a ROS publisher for R is more complex and was therefore outsourced. The header file contains the implementation of a specific publisher class developed for R. Other functions defined within the 'cpp'-file, only give access to this specific R-publisher and allow the modification of its settings, such as topic, message type, etc., but also to alter the content of its message buffer (derived from topic_tools).
    - 'SubscribeR.h/cpp': The subscriber was developed similarly to the implementation of the publisher. The header filer contains the implementation of a specific R subscriber. Due to the fact that callbacks cannot be defined within R, every SubscribeR object also contains a callback method, which comes into play when ros::SpinOnce() is activated. It receives and stores every new message. Other functions defined within these files give access to the subscriber and its message buffer.
    - 'BagR.h/cpp' and 'ParamR.h/cpp': As the names suggest, these files contain the functions to read out bag-files and give access to the parameter server.
    - 'rosR_wrap.cxx': This file contains the generated wrapper and conversion functionality for C++, which is based on **Rcpp**.

SWIG by example

The following two functions are taken from 'ParamR.cpp'. The first function is used to retrieve string parameters from the parameter server, similar functions were also defined to retrieve double, boolean, and integer values. The second function identifies the type of the parameter, returning the type as a string and the string NULL, if the parameter does not exist.

```
char* rrosGetParamString(
   ros::NodeHandle* handle,
   char* param)
{
  std::string val;
  handle->getParam(param, val);
  return const_cast<char*>(val.c_str());
}
```

```
char* rrosGetParamType(ros::NodeHandle* h,
    char* p)//(==param) requested parameter
{
  bool b; int i; double d; std::string s;
  if(h->getParam(p, b)) return "logical";
  if(h->getParam(p, i)) return "integer";
  ...
  return "NULL"; }
```

After SWIG2.0 did its magic, these C++ functions are also callable from R (with their pervious C++ function names). The API functions ros.ParamGet and ros.ParamType are simple workarounds (defined in 'rosR/lib/ros.R') that hide the different data types and therefore different C++ function calls:

```
ros.ParamGet <- function(param) {
  p <- param; h <-rros_node;
  type <- ros.ParamType(param)
  if (type == "logical") {
    return(rrosGetParamBoolean(h, p))
  } else if (type == "integer") {
    return(rrosGetParamInteger(h, p))
  } else if (type == "double") {
    return(rrosGetParamDouble(h, p))
  } else if (type == "character") {
    return(rrosGetParamString(h, p))
  } else { return(NULL) } }
```

```
# "rros_node" is a pointer to the ROS node-
# handle, which was created during the ini-
# tialization by "ros.Init()".
# It is stored as a global variable and re-
# quired by most of the rosR API functions.

ros.ParamType <- function(param) {
  p <- param; h <-rros_node;
  type <- rrosGetParamType(h, p)
  if (type == "NULL") {
    return(NULL) }
  return(type) }
```

The other functions of our API were also implemented in the same manner (cf. 'ros.R'), while announcing a subscriber or a publisher (and also the initialization with `ros.Init`) also generate specified objects. The wrapper functions for writing and reading out messages therefore always require the pointer to these objects, to get access to their public class methods.

Summary

We have developed the first interface for ROS in R and demonstrated its utility as well as its main concepts, which represents a fruitful combination of the (interdisciplinary) world of robotics with the world of statistics. On the one hand, it gives ROS developers full access to the huge amount of R algorithms and functionality for analyzing, visualizing, combining and filtering data. And on the other hand, it will probably open up a new branch to the R community, by giving online access to real hardware devices and their data, in contrast to the traditional offline data analysis.

But, there is still a lot of work to do. The next extensions should cover the integration of services, which could be used for parameter fitting, clustering, machine learning, or pattern matching. Furthermore, it would be beneficial to develop methods in R that would enable callback mechanisms or multi-threading, to overcome polling for new messages.

Acknowledgements

This work is funded and supported by the German Ministry of Education and research within the project ViERforES-II (grant no. 01IM10002B) and by the EU FP7-ICT program under the contract number 288195 “Kernel-based ARchitecture for safetY-critical cONtrol” (KARYON).

Bibliography

A. Blasdel. rqt_console: A GUI plugin for displaying and filtering ROS messages, 2013. URL `http://wiki.ros.org/rqt_console`. [Online; accessed 16-December-2013]. [p122]

K. Conley. rostopic: Command-line tool for displaying debug information, 2013a. URL `http://wiki.ros.org/rostopic`. [Online; accessed 16-December-2013]. [p126]

K. Conley. rxgraph: Command-line tool for visualizing a ROS computation graph, 2013b. URL `http://www.ros.org/wiki/rxgraph`. [Online; accessed 16-December-2013]. [p121]

K. Conley and T. Foote. rosmsg: Command-line tool for displaying information about ROS message types, 2013. URL `http://wiki.ros.org/rosmsg`. [Online; accessed 16-December-2013]. [p124]

A. Dietrich, S. Zug, and J. Kaiser. Detecting external measurement disturbances based on statistical analysis for smart sensors. In *Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE)*, pages 2067–2072, July 2010. [p118]

D. Eddelbuettel and R. François. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. URL `http://www.jstatsoft.org/v40/i08/`. [p120]

T. Field, J. Leibs, and J. Bowman. rosbag: A set of tools for recording from and playing back to ROS topics, 2013. URL `http://wiki.ros.org/rosbag`. [Online; accessed 16-December-2013]. [p121, 127]

T. Foote. rosmake: A ROS dependency aware build tool, 2013. URL `http://wiki.ros.org/rosmake`. [Online; accessed 16-December-2013]. [p120]

B. Gerkey, M. Quigley, and D. Thomas. rospack: ROS package management tool, 2013. URL `http://docs.ros.org/independent/api/rospkg/html/rospack.html`. [Online; accessed 16-December-2013]. [p120]

J. Kramer and M. Scheutz. Development environments for autonomous mobile robots: A survey. *Autonomous Robots*, 22:101–132, 2007. [p118]

J. Leibs and T. Kruse. rosrun: Allows you to run an executable in an arbitrary package from anywhere without having to give its full path, 2013. URL `http://wiki.ros.org/rosbash#rosrun`. [Online; accessed 16-December-2013]. [p122]

M. Quigley and B. Gerkey. topic_tools: Directing, throttling, selecting, and otherwise messing with ROS topics at a meta level, 2013. URL `http://wiki.ros.org/topic_tools`. [Online; accessed 16-December-2013]. [p127]

M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng. ROS: An open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009. [p118]

M. Quigley, J. Faust, B. Gerkey, and T. Straszheim. roscpp: C++ implementation of ROS, 2013. URL `http://wiki.ros.org/roscpp`. [Online; accessed 16-December-2013]. [p118, 127]

P. H. Salus. *A Quarter Century of Unix*. Addison-Wesley Longman, 1994. [p118]

Sharp Cooperation. *Sharp GP2D120 – Optoelectronic Device*, 2007. URL `http://www.sharpsma.com/webfm_send/1205`. [p120, 123, 124]

SWIG-2.0. Documentation, 2013. URL `http://www.swig.org/Doc2.0/SWIGDocumentation.html`. [Online; accessed 21-July-2013]. [p119]

Wikipedia. Comparison of statistical packages – Wikipedia, The Free Encyclopedia, 2013. URL `http://en.wikipedia.org/wiki/Comparison_of_statistical_packages`. [Online; accessed 21-July-2013]. [p118]

André Dietrich, Sebastian Zug, and Jörg Kaiser
Department of Distributed Systems
Embedded Systems and Operating Systems Working Group
Otto-von-Guericke-Universität Magdeburg
Universitätsplatz 2, 39106 Magdeburg, Germany
`{dietrich, zug, kaiser}@ivs.cs.uni-magdeburg.de`

On Sampling from the Multivariate *t* Distribution

by Marius Hofert

Abstract The multivariate normal and the multivariate *t* distributions belong to the most widely used multivariate distributions in statistics, quantitative risk management, and insurance. In contrast to the multivariate normal distribution, the parameterization of the multivariate *t* distribution does not correspond to its moments. This, paired with a non-standard implementation in the R package **mvtnorm**, provides traps for working with the multivariate *t* distribution. In this paper, common traps are clarified and corresponding recent changes to **mvtnorm** are presented.

Introduction

A supposedly simple task in statistics courses and related applications is to generate random variates from a multivariate *t* distribution in R. When teaching such courses, we found several fallacies one might encounter when sampling multivariate *t* distributions with the well-known R package **mvtnorm**; see Genz et al. (2013). These fallacies have recently led to improvements of the package ($\geq$ 0.9-9996) which we present in this paper[1]. To put them in the correct context, we first address the multivariate normal distribution.

The multivariate normal distribution

The *multivariate normal distribution* can be defined in various ways, one is with its stochastic representation

$$\boldsymbol{X} = \boldsymbol{\mu} + A\boldsymbol{Z}, \tag{1}$$

where $\boldsymbol{Z} = (Z_1, \dots, Z_k)$ is a k-dimensional random vector with Z_i, $i \in \{1, \dots, k\}$, being independent standard normal random variables, $A \in \mathbb{R}^{d\times k}$ is an (d,k)-matrix, and $\boldsymbol{\mu} \in \mathbb{R}^d$ is the mean vector. The covariance matrix of $\boldsymbol{X}$ is $\Sigma = AA^\top$ and the distribution of $\boldsymbol{X}$ (that is, the d-dimensional *multivariate normal distribution*) is determined solely by the mean vector $\boldsymbol{\mu}$ and the covariance matrix Σ; we can thus write $\boldsymbol{X} \sim \mathrm{N}_d(\boldsymbol{\mu}, \Sigma)$.

In what follows, we assume $k = d$. If Σ is positive definite (thus has full rank and is therefore invertible), $\boldsymbol{X}$ has density

$$f_{\boldsymbol{X}}(\boldsymbol{x}) = \frac{1}{(2\pi)^{d/2}\sqrt{\det\Sigma}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right), \quad \boldsymbol{x} \in \mathbb{R}^d.$$

Contours of equal density are ellipsoids; all so-called elliptical distributions which admit a density share this property.

A positive definite (semi-definite) matrix $\Sigma \in \mathbb{R}^{d\times d}$ can be written as

$$\Sigma = LL^\top \tag{2}$$

for a lower triangular matrix L with $L_{jj} > 0$ ($L_{jj} \geq 0$) for all $j \in \{1, \dots, d\}$. L is known as the *Cholesky factor* of the *Cholesky decomposition* (2).

The stochastic representation (1), together with the Cholesky decomposition of Σ, allows for a direct sampling strategy of multivariate normal distributions $\mathrm{N}_d(\boldsymbol{\mu}, \Sigma)$, which can easily be implemented in R as follows.

```
> ## Setup
> mu <- 1:2 # mean vector of X
> Sigma <- matrix(c(4, 2, 2, 3), ncol=2) # covariance matrix of X
> n <- 1000 # sample size
> d <- 2 # dimension
> ## Step 1: Compute the Cholesky factor of Sigma
> L <- t(chol(Sigma)) # t() as chol() returns an upper triangular matrix
> ## Step 2: Generate iid standard normal random variates
```

[1]The accompanying R script may be obtained from the author upon request.

```
> set.seed(271) # set seed for reproducibility
> Z <- matrix(rnorm(n*d), nrow=d, ncol=n) # (d,n)-matrix
> ## Step 3: Reconstruct the stochastic representation
> X <- mu + L %*% Z # (d,n)-matrix of realizations N_d(mu, Sigma)
```

This idea for sampling $\boldsymbol{X} \sim \mathrm{N}_d(\boldsymbol{\mu}, \Sigma)$ is available in the R package **mvtnorm** as follows:

```
> require(mvtnorm)
> set.seed(271)
> X. <- rmvnorm(n, mean=mu, sigma=Sigma, method="chol") # (n,d)-matrix
> stopifnot(identical(t(X), X.)) # check equality of the random numbers
```

The default method (`method="eigen"`) utilizes the eigendecomposition of Σ, which also applies if some eigenvalues are 0. The function `mvrnorm()` of the recommended R package **MASS** provides the same approach; see Venables and Ripley (2013) or (Ripley, 1987, pp. 98). Note however, that although the internally drawn independent standard normal random variates are identical, the two algorithms compute different matrices A such that $AA^\top = \Sigma$ and thus do not lead to identical $\mathrm{N}_d(\boldsymbol{\mu}, \Sigma)$ random variates.

```
> require(MASS)
> X.. <- mvrnorm(n, mu=mu, Sigma=Sigma) # (n,d)-matrix
```

The multivariate t distribution

The left-hand side of Figure 1 displays 1000 log-returns of daily stock prices for BMW and Siemens in the decade from 1985-01-02 to 1994-12-30; the data is available in the R package **evir** (Pfaff, 2012). This time period includes various historically "extreme" events such as the stock market crash 1987 known as "Black Monday" (1987-10-19), one of the Monday demonstrations in Leipzig (1989-10-16), and the August Putsch attempt against Mikhail Gorbachev (1991-08-19).

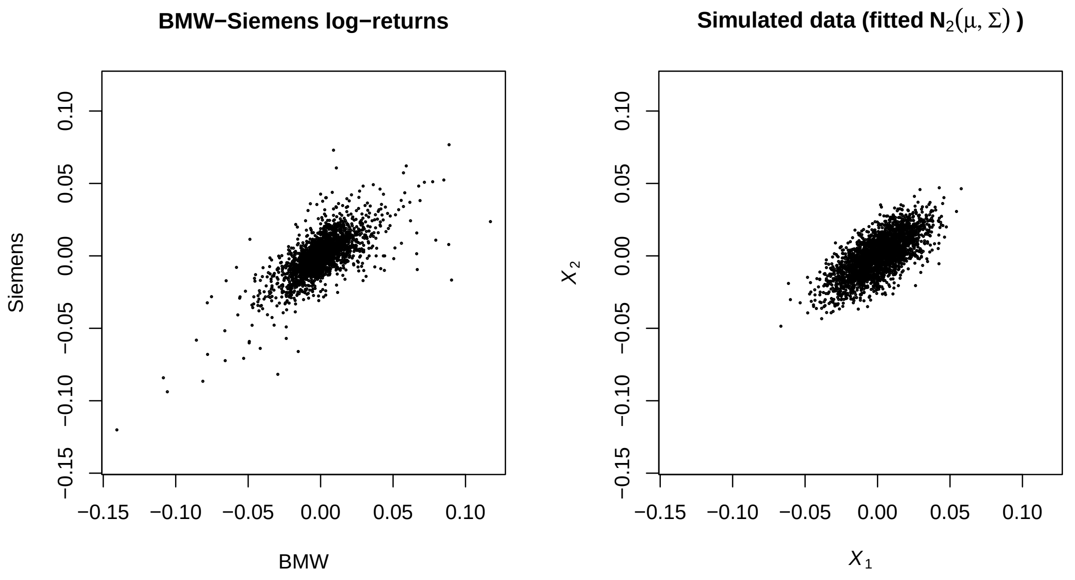


Figure 1: Log-returns for daily (BMW, Siemens) data from 1985-01-02 to 1994-12-30 (left). Correspondingly many simulated data points from a fitted bivariate normal distribution (right).

A comparison with simulated data of the same sample size from a fitted bivariate normal distribution on the right-hand side of Figure 1 shows that the bivariate normal distribution is not an adequate model here to account for such (joint) "extreme" events (in the upper right or lower left corner of the bivariate distribution); this can also be checked with formal statistical or graphical goodness-of-fit tests. The bivariate t distribution typically captures such events better (mathematically speaking, it is able to capture "tail dependence") and has gained popularity in modeling such events, for example, in quantitative risk management applications.

Definition and density

The *multivariate t distribution with ν degrees of freedom* can be defined by the stochastic representation

$$\boldsymbol{X} = \boldsymbol{\mu} + \sqrt{W} A \boldsymbol{Z}, \tag{3}$$

where $W = \nu/\chi^2_\nu$ (χ^2_ν is informally used here to denote a random variable following a chi-squared distribution with $\nu > 0$ degrees of freedom) is independent of $\boldsymbol{Z}$ and all other quantities are as in (1).

By introducing the additional random factor $\sqrt{W}$, the multivariate t distribution with ν degrees of freedom (denoted by $t_\nu(\boldsymbol{\mu}, \Sigma)$) is more flexible than the multivariate normal distribution (which can be recovered by taking the limit $\nu \to \infty$) especially in the tails which are heavier for $t_\nu(\boldsymbol{\mu}, \Sigma)$ than for $\mathrm{N}_d(\boldsymbol{\mu}, \Sigma)$. The density of $\boldsymbol{X} \sim t_\nu(\boldsymbol{\mu}, \Sigma)$ is given by

$$f_{\boldsymbol{X}}(\boldsymbol{x}) = \frac{\Gamma\left(\frac{\nu+d}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)(\pi\nu)^{d/2}\sqrt{\det\Sigma}} \left(1 + \frac{(\boldsymbol{x}-\boldsymbol{\mu})^\top \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}{\nu}\right)^{-\frac{\nu+d}{2}}, \quad \boldsymbol{x} \in \mathbb{R}^d. \tag{4}$$

As for the multivariate normal distribution, the density (4) has ellipsoidal level sets and thus belongs to the class of elliptical distributions.

In R

As the CRAN Task View "Distributions" reveals, the R packages **mvtnorm** and **mnormt** (see Azzalini, 2012, for the latter) provide functions for drawing random variates from the multivariate normal and t distributions. The former is the most widely used among all non-recommended packages (measured by Reverse Depends as of August 5, 2012; see Eddelbuettel, 2012). In what follows, we present and discuss changes to **mvtnorm** ($\geq$ 0.9-9996) which were inspired by the corresponding fallacies. Afterwards, we will briefly address **mnormt**.

Fallacies when sampling the multivariate t distribution

The left-hand side of Figure 2 shows 2608 simulated vectors from a bivariate t distribution fitted to the BMW–Siemens log-return data. The parameters are estimated as follows, utilizing the R package **QRM** (Pfaff and McNeil, 2012).

```
> require(QRM)
> fit <- fit.mst(X, method = "BFGS") # fit a multivariate t distribution
> mu <- fit$mu # estimated location vector
> Sigma <- as.matrix(fit$Sigma) # estimated scale matrix
> nu <- fit$df # estimated degrees of freedom
```

In comparison to the sample from the multivariate normal distribution on the right-hand side of Figure 1, the multivariate t distribution shows significantly heavier tails. This is also indicated by the estimated degrees of freedom parameter $\nu \approx 3.02$.

The task

We now turn to the task of generating vectors of random variates from a multivariate t distribution with ν degrees of freedom, that is, generating samples such as shown on the left-hand side of Figure 2. We assume

$$\boldsymbol{\mu} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} 4 & 2 \\ 2 & 3 \end{pmatrix}, \quad \nu = 3, \tag{5}$$

and try to generate $n = 1000$ samples.

Fallacy 1: Assuming it to work as expected

The obvious generalization of drawing n samples from $\mathrm{N}_d(\boldsymbol{\mu}, \Sigma)$ via `rmvnorm(n,mean=mu,sigma=Sigma)` (with `n` being n, `mu` being $\boldsymbol{\mu}$, and `Sigma` being Σ) to $t_\nu(\boldsymbol{\mu}, \Sigma)$ would be to use `rmvt(n,mean=mu,sigma=Sigma,df=nu)` (with `nu` being ν):

```
> require(mvtnorm)
> n <- 1000
```

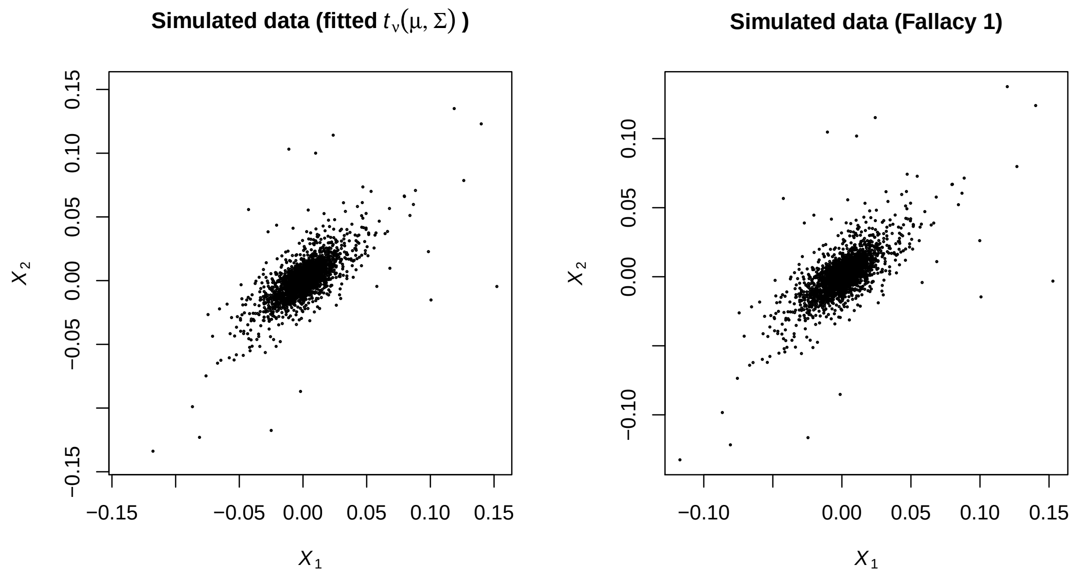


Figure 2: Simulated data (sample size 2608) from a fitted bivariate t distribution ($\nu \approx 3.02$) (left). Simulated data (sample size 2608) from (5) with the approach described in Fallacy 1 (7) (right).

```
> mu <- 1:2
> Sigma <- matrix(c(4, 2, 2, 3), ncol=2)
> nu <- 3
> set.seed(271)
> X1 <- try(rmvt(n, mean=mu, sigma=Sigma, df=nu)) # error; 'mean' not allowed anymore
```

In **mvtnorm** ($\geq$ 0.9-9996), this generates an error and is thus not allowed.

To understand why it is dangerous not to throw an error in this situation, let us look at the mean. Theoretically, the mean of $\boldsymbol{X} \sim t_\nu(\boldsymbol{\mu}, \Sigma)$ is given by

$$\mathbb{E}[\boldsymbol{X}] = \mathbb{E}[\boldsymbol{\mu} + \sqrt{W}A\boldsymbol{Z}] = \boldsymbol{\mu} + \mathbb{E}[\sqrt{W}]A\mathbb{E}[\boldsymbol{Z}] = \boldsymbol{\mu}, \tag{6}$$

where we used that W and $\boldsymbol{Z}$ are independent, $\mathbb{E}[\boldsymbol{Z}] = \boldsymbol{0}$, and that $\nu > 1$ (so that $\mathbb{E}[\sqrt{W}]$ exists). In previous versions of `rmvt()`, a specified argument `mean` was passed to the internal call of `rmvnorm()` via the ellipsis "...". This implies that one actually generated a sample from

$$\boldsymbol{X} = \sqrt{W}\boldsymbol{Y}, \tag{7}$$

where $\boldsymbol{Y} \sim \mathrm{N}_d(\boldsymbol{\mu}, \Sigma)$. The expectation of $\boldsymbol{X}$ in this case is

$$\mathbb{E}[\boldsymbol{X}] = \mathbb{E}[\sqrt{W}]\boldsymbol{\mu}$$

instead of $\boldsymbol{\mu}$. $\sqrt{W}$ has distribution function $F_{\sqrt{W}}(x) = 1 - F_{\chi^2_\nu}(\nu/x^2)$ with density $f_{\sqrt{W}}(x) = 2\nu f_{\chi^2_\nu}(\nu/x^2)/x^3$. Using the fact that a χ^2_ν distribution is a $\Gamma(\nu/2, 1/2)$ distribution with density $f_{\Gamma(\nu/2,1/2)}(x) = (1/2)^{\nu/2}x^{\nu/2-1}\exp(-x/2)/\Gamma(\nu/2)$, a rather tedious than complicated calculation shows that

$$\mathbb{E}[\sqrt{W}] = \sqrt{\frac{\nu}{2}}\frac{\Gamma((\nu-1)/2)}{\Gamma(\nu/2)}$$

and thus that $\mathbb{E}[\boldsymbol{X}] \approx (1.38, 2.76)$ rather than the required $\boldsymbol{\mu} = (1, 2)$. Table 1 gives an intuition for how fast $\mathbb{E}[\sqrt{W}]$ converges to 1 for large ν. In financial applications, one typically has ν values between 3 and 5 which implies values of $\mathbb{E}[\sqrt{W}]$ between 1.3820 and 1.1894, respectively.

ν	1.0080	1.0794	1.7757	8.5417	76.0418	751.0417
$\mathbb{E}[\sqrt{W}]$	101	11	2	1.1	1.01	1.001

Table 1: Values of ν for $\mathbb{E}[\sqrt{W}] \in \{1 + 10^i : i \in \{2, 1, 0, -1, -2, -3\}\}$.

It follows from (1) and (7) that previous versions of `rmvt()` with specified argument `mean` actually sampled from a random vector $\boldsymbol{X}$ with stochastic representation

$$\boldsymbol{X} = \sqrt{W}\boldsymbol{\mu} + \sqrt{W}A\boldsymbol{Z}. \tag{8}$$

The distribution of such $\boldsymbol{X}$ belongs to the class of *normal mean-variance mixtures*; see (McNeil et al., 2005, Section 3.2.2). In general, such distributions are not elliptical distributions anymore. By looking at the source code of `rmvt()`, we can still mimic this previous behavior of `rmvt(n,mean=mu,sigma=Sigma,df=nu)` by

```
> set.seed(271)
> ## exactly the random variates drawn by rmvt(n, mean=mu, sigma=Sigma, df=nu)
> ## in versions of mvtnorm before 0.9-9996:
> X12 <- rmvt(n, sigma=Sigma, df=nu, delta=mu, type="Kshirsagar")
> colMeans(X12) # => wrong (sample) mean
```

```
[1] 1.5380 2.7955
```

The result is shown on the right-hand side of Figure 2. In contrast to many other sampling functions in R (even including **mnormt**'s `rmt()`), `rmvt()` does not have an argument `mean` and previous versions of `rmvt()` thus generated random variates from (8) instead if this argument was provided.

Remark 1
- *As we can see from the last chunk,* `rmvt()` *with* `type="Kshirsagar"` *specifically allows to sample* (8). *For other applications of* `type="Kshirsagar"`, *see* `?rmvt`.
- *We saw in* (6) *that* $\boldsymbol{\mu}$ *is only the mean of* $\boldsymbol{X}$ *if* $\nu > 1$. *The parameter* $\boldsymbol{\mu}$ *of* $t_\nu(\boldsymbol{\mu}, \Sigma)$ *is therefore referred to as* location vector *(as opposed to "mean vector").*

Fallacy 2: Vector vs matrix

To properly specify the location vector, R users are often tempted to do the following:

```
> X2 <- mu + rmvt(n, sigma=Sigma, df=nu)
```

The problem with this approach is not visible for the human eye here! To make it more pronounced, let us blow up the location vector:

```
> set.seed(271)
> X2 <- 20*mu + rmvt(n, sigma=Sigma, df=nu)
```

The left-hand side of Figure 3 reveals the problem. Indeed, we added the *vector* `20*mu` to the *matrix* returned by `rmvt()`. R solves this problem by sufficiently often repeating the vector elements to obtain a matrix of the same size such that addition makes sense.

```
> head(matrix(20*mu, nrow=n, ncol=d))
```

```
     [,1] [,2]
[1,]   20   20
[2,]   40   40
[3,]   20   20
[4,]   40   40
[5,]   20   20
[6,]   40   40
```

As we can see (and more experienced R users know this fact well), matrices are filled and worked on column-wise. So every second sample has a different mean vector (alternating between (20, 20) and (40, 40)). We thus sampled a mixture of two t distributions and again have left the class of elliptical distributions. The left-hand side of Figure 3 clearly indicates this by showing the two clouds centered around the two mean vectors. This problem is virtually impossible to detect here without the scaling factor (and thus harbors risk of being overlooked).

In order to take care of the correct mean, there are several possibilities, some are:

```
> set.seed(271)
> X21 <- matrix(mu, nrow=n, ncol=d, byrow=TRUE) + rmvt(n, sigma=Sigma, df=nu)
> set.seed(271)
> X22 <- rep(mu, each=n) + rmvt(n, sigma=Sigma, df=nu)
> set.seed(271)
> X23 <- sweep(rmvt(n, sigma=Sigma, df=nu), MARGIN=2, STATS=mu, FUN="+")
> stopifnot(identical(X21, X22),
            identical(X21, X23)) # check equality of the random numbers
```

The last approach is implemented in `rmvt()` in terms of the argument `delta` (if the argument `type` attains its default `"shifted"`):

```
> set.seed(271)
> X24 <- rmvt(n, sigma=Sigma, df=nu, delta=mu)
> stopifnot(identical(X21, X24))
```

Fallacy 3: The meaning of Σ and `sigma`

After having taken care of the mean vector $\boldsymbol{\mu}$, let us now consider Σ.

```
> set.seed(271)
> X3 <- rmvt(n, sigma=Sigma, df=nu, delta=mu)
> cov(X3)
```

```
       [,1]   [,2]
[1,] 9.8843 4.9204
[2,] 4.9204 7.6861
```

As we see, the sample covariance matrix is not close to Σ as specified in (5).

For $\boldsymbol{X} \sim \mathrm{N}_d(\boldsymbol{\mu}, \Sigma)$ it is easy to see that $\mathbb{E}[\boldsymbol{X}] = \boldsymbol{\mu}$ and $\mathrm{Cov}[\boldsymbol{X}] = \Sigma$, so $\boldsymbol{\mu}$ and Σ are indeed the mean vector and covariance matrix of $\boldsymbol{X}$, respectively. As we have already seen for $\boldsymbol{\mu}$, this is not necessarily true anymore for $t_\nu(\boldsymbol{\mu}, \Sigma)$ due to the additional random factor $\sqrt{W}$ in the stochastic representation (3). The same applies to the covariance matrix of $\boldsymbol{X}$. It follows from (3) that if $\mathbb{E}[W] < \infty$, we have

$$\mathrm{Cov}[\boldsymbol{X}] = \mathrm{Cov}[\sqrt{W}A\boldsymbol{Z}] = \mathbb{E}[(\sqrt{W}A\boldsymbol{Z})(\sqrt{W}A\boldsymbol{Z})^\top] = \mathbb{E}[W]\,\mathrm{Cov}[A\boldsymbol{Z}] = \mathbb{E}[W]\Sigma.$$

It is a basic exercise to show that $W = \nu/\chi^2_\nu$ implies that $\mathbb{E}[W] = \frac{\nu}{\nu-2}$. Therefore, the covariance matrix of $\boldsymbol{X}$ only exists if $\nu > 2$ in which case it is $\mathrm{Cov}[\boldsymbol{X}] = \frac{\nu}{\nu-2}\Sigma$, not Σ. For this reason, Σ is referred to as *scale* (or *dispersion*) *matrix* in order to distinguish it from the covariance matrix of $\boldsymbol{X}$ (which does not have to exist). In our task (5) the covariance matrix of $\boldsymbol{X}$ is 3Σ which is roughly what we see above (and which can be confirmed by a larger sample size). `X3` (displayed on the right-hand side of Figure 3) therefore shows a sample from the correct distribution as specified by (5); note that the same construction principle has been used to create the left-hand side of Figure 2.

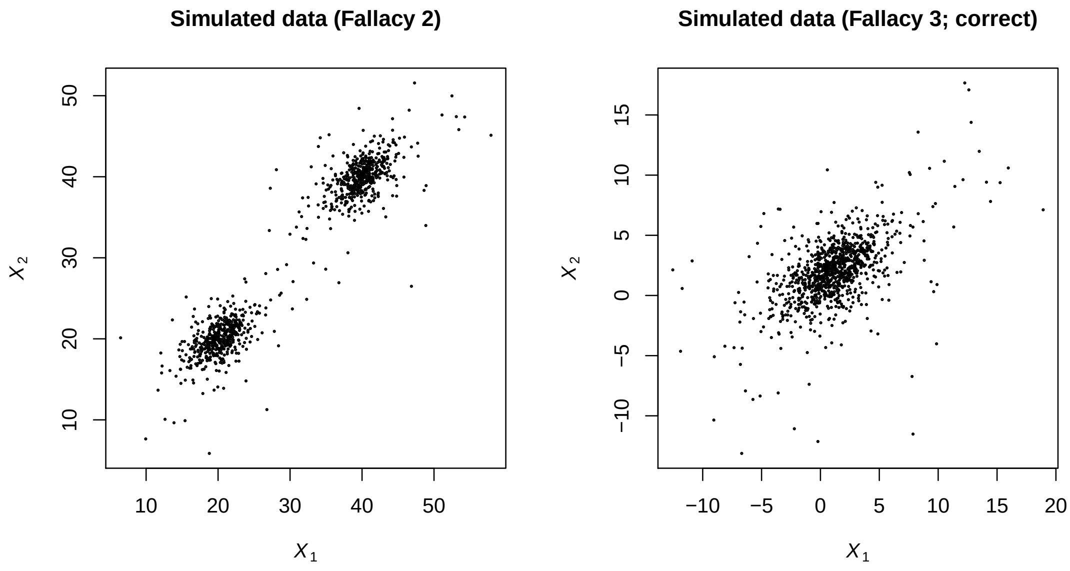


Figure 3: Simulated data from the approach described in Fallacy 2 (left) and Fallacy 3 (right).

Finally, let us mention that if $\nu > 2$, then

$$\mathrm{Cor}[\boldsymbol{X}] = P = (\rho_{ij})_{i,j\in\{1,\dots,d\}}, \quad \text{where } \rho_{ij} = \frac{\mathrm{Cov}[X_i, X_j]}{\sqrt{\mathrm{Var}[X_i]\,\mathrm{Var}[X_j]}}.$$

Let $\Sigma = (\sigma_{ij})_{i,j\in\{1,\dots,d\}}$. Since

$$\begin{aligned}\operatorname{Cov}[X_i, X_j] &= \mathbb{E}[W]\sigma_{ij}, \quad i,j \in \{1,\dots,d\},\\ \operatorname{Var}[X_k] &= \mathbb{E}[(\sqrt{W}(A\boldsymbol{Z})_k)^2] = \mathbb{E}[W]\operatorname{Var}[(A\boldsymbol{Z})_k] = \mathbb{E}[W]\sigma_{kk}, \quad k \in \{1,\dots,d\},\end{aligned}$$

where $(A\boldsymbol{Z})_k$ denotes the kth row of $A\boldsymbol{Z}$, we obtain

$$P_{ij} = \frac{\mathbb{E}[W]\sigma_{ij}}{\sqrt{\mathbb{E}[W]\sigma_{ii}\mathbb{E}[W]\sigma_{jj}}} = \frac{\sigma_{ij}}{\sqrt{\sigma_{ii}\sigma_{jj}}}.$$

If $\nu > 2$, we see that although the covariance matrix $\operatorname{Cov}[\boldsymbol{X}]$ is not equal to the scale matrix Σ, the correlation matrix $\operatorname{Cor}[\boldsymbol{X}]$ is equal to the correlation matrix P corresponding to the scale matrix Σ. This can also be verified numerically:

```
> set.seed(271)
> ## sample correlation matrix of a t3 sample with scale matrix Sigma
> cor(rmvt(1e6, sigma=Sigma, df=3, delta=mu))

        [,1]    [,2]
[1,] 1.00000 0.57667
[2,] 0.57667 1.00000

> ## correlation matrix corresponding to Sigma
> cov2cor(Sigma)

        [,1]    [,2]
[1,] 1.00000 0.57735
[2,] 0.57735 1.00000
```

Remark 2 *The user should also be aware of the fact that* `rmvt(n,delta=mu,sigma=Sigma,df=0,...)` *is equivalent to* `rmvnorm(n,mean=mu,sigma=Sigma,...)`*. This is counter-intuitive as the multivariate normal distribution arises as $\nu \to \infty$, not $\nu \to 0+$. This might be problematic for very small degrees of freedom parameters ν due to truncation to 0. Note that* **mvtnorm** *($\geq$ 0.9-9996) now also allows to specify* `df=Inf` *for the (same) limiting multivariate normal distribution. The case* `df=0` *remains for backward compatibility.*

Comparison with mnormt

The R package **mnormt** provides the function `rmt()` for sampling from the multivariate t distribution. The call `rmt(n,mean=mu,S=Sigma,df=nu)` provides the correct answer to task (5). Note that `rmt()` has an argument `mean`, but actually means the location vector (see `?rmt`). Furthermore, there is a subtle difference between **mnormt** and **mvtnorm**. **mnormt**'s `rmnorm()` uses a Cholesky decomposition of Σ. Even by starting with the same seed and calling **mvtnorm**'s `rmvt()` with `method="chol"`, the vectors of random variates generated by `rmt(n,mean=mu,S=Sigma,df=nu)` and those by `rmvt(n,sigma=Sigma,df=nu, delta=mu,method="chol")` are not identical. The reason for this is that the order in which the normal and the χ^2_ν distributed random variates are generated differs for `rmt()` and `rmvt()`.

Summary and conclusion

Table 2 collects the various calls of `rmvt()` and corresponding stochastic representations of $\boldsymbol{X}$ we encountered.

To summarize, let $\boldsymbol{X} \sim t_\nu(\boldsymbol{\mu}, \Sigma)$. Then:

1. The location vector $\boldsymbol{\mu}$ is not equal to $\mathbb{E}[\boldsymbol{X}]$ unless $\nu > 1$ and thus $\mathbb{E}[\boldsymbol{X}]$ exists. The scale matrix Σ is *a* covariance matrix but not equal to *the* covariance matrix of $\boldsymbol{X}$.
2. $\boldsymbol{X}$ can be sampled via `rmvt(n,sigma=Sigma,df=nu,delta=mu)`, where `n` is the sample size n, `mu` the location vector $\boldsymbol{\mu}$, `Sigma` the scale matrix Σ, and `nu` the degrees of freedom parameter ν; this holds for all $\nu > 0$ (watch out for very small ones, though).
3. The argument `sigma` of `rmvt()` denotes the scale matrix Σ of $\boldsymbol{X}$. If the scale matrix Σ is standardized, it is *a* correlation matrix, but not necessarily *the* correlation matrix of $\boldsymbol{X}$ (the latter does not have to exist). Only if $\nu > 2$, $\operatorname{Cor}[\boldsymbol{X}]$ exists and equals the correlation matrix corresponding to the scale matrix Σ (which can be computed via `cov2cor()`).

Call	Stochastic representation of $\boldsymbol{X}$
`X1  <- rmvt(n, mean=mu, sigma=Sigma, df=nu);`	gives an error
`X12 <- rmvt(n, sigma=Sigma, df=nu, delta=mu, type="Kshirsagar");`	$\boldsymbol{X} = \sqrt{W}(\boldsymbol{\mu} + A\boldsymbol{Z})$
`X2  <- mu + rmvt(n, sigma=Sigma, df=nu);`	mixture of two t distributions (see text)
`X21 <- matrix(mu, nrow=n, ncol=d, byrow=TRUE) + rmvt(n, sigma=Sigma, df=nu);`	as `X3`
`X22 <- rep(mu, each=n) + rmvt(n, sigma=Sigma, df=nu);`	as `X3`
`X23 <- sweep(rmvt(n, sigma=Sigma, df=nu), MARGIN=2, STATS=mu, FUN="+");`	as `X3`
`X24 <- rmvt(n, sigma=Sigma, df=nu, delta=mu);`	as `X3`
`X3  <- rmvt(n, sigma=Sigma, df=nu, delta=mu);`	$\boldsymbol{X} = \boldsymbol{\mu} + \sqrt{W}A\boldsymbol{Z} \sim t_\nu(\boldsymbol{\mu}, \Sigma)$

Table 2: Calls of `rmvt()` and corresponding stochastic representations of $\boldsymbol{X}$.

Acknowledgments

I would like to thank Martin Mächler (ETH Zürich) and Gabriel Doyon (ETH Zürich) for valuable feedback on an early version of this work. Furthermore, I would like to thank the Associate Editor Bettina Grün (Johannes Kepler Universität Linz) and two anonymous reviewers for their feedback during the review process.

Bibliography

A. Azzalini. *mnormt: The Multivariate Normal and t Distributions*, 2012. URL http://CRAN.R-project.org/package=mnormt. R package version 1.4.5. [p132]

D. Eddelbuettel. Counting CRAN package Depends, Imports and LinkingTo, 2012. URL http://dirk.eddelbuettel.com/blog/2012/08/05/. [p132]

A. Genz, F. Bretz, T. Miwa, X. Mi, F. Leisch, F. Scheipl, and T. Hothorn. *mvtnorm: Multivariate Normal and t Distributions*, 2013. URL http://CRAN.R-project.org/package=mvtnorm. R package version 0.9-9996. [p130]

A. J. McNeil, R. Frey, and P. Embrechts. *Quantitative Risk Management: Concepts, Techniques, Tools*. Princeton University Press, 2005. [p134]

B. Pfaff. *evir: Extreme Values in R*, 2012. URL http://CRAN.R-project.org/package=evir. R package version 1.7-3. [p131]

B. Pfaff and A. McNeil. *QRM: Provides R-Language Code to Examine Quantitative Risk Management Concepts*, 2012. URL http://CRAN.R-project.org/package=QRM. R package version 0.4-8. [p132]

B. D. Ripley. *Stochastic Simulation*. John Wiley & Sons, 1987. [p131]

W. N. Venables and B. D. Ripley. *MASS: Support Functions and Datasets for Venables and Ripley's MASS*, 2013. URL http://CRAN.R-project.org/package=MASS. R package version 7.3.26. [p131]

Marius Hofert
RiskLab, Department of Mathematics,
ETH Zürich, 8092 Zürich,
Switzerland.
marius.hofert@math.ethz.ch

betategarch: Simulation, estimation and forecasting of Beta-Skew-t-EGARCH Models

by Genaro Sucarrat

Abstract This paper illustrates the usage of the **betategarch** package, a package for the simulation, estimation and forecasting of Beta-Skew-t-EGARCH models. The Beta-Skew-t-EGARCH model is a dynamic model of the scale or volatility of financial returns. The model is characterised by its robustness to jumps or outliers, and by its exponential specification of volatility. The latter enables richer dynamics, since parameters need not be restricted to be positive to ensure positivity of volatility. In addition, the model also allows for heavy tails and skewness in the conditional return (i.e. scaled return), and for leverage and a time-varying long-term component in the volatility specification. More generally, the model can be viewed as a model of the scale of the error in a dynamic regression.

Introduction

It is well known that financial returns are characterised by volatility clustering: Large returns in absolute value are likely to be followed by other large returns in absolute value, and small returns in absolute value are likely to be followed by other small returns in absolute value. This characteristic is usually modelled by specifications in the Autoregressive Conditional Heteroscedasticity (ARCH) class of models by Engle (1982). If y_t denotes the financial return at t such that

$$y_t = \sigma_t \varepsilon_t, \qquad \varepsilon_t \sim IID(0, \sigma_\varepsilon^2), \qquad \sigma_t^2 = h(\sigma_{t-1}, y_{t-1}, \ldots), \qquad t = 1, 2, \ldots,$$

then the scale or volatility $\sigma_t > 0$ is said to follow an ARCH process. Arguably, the most popular ARCH specification is the first order Generalised ARCH (GARCH) of Bollerslev (1986), where σ_t^2 is modelled in an ARMA(1,1)-like manner, $\sigma_t^2 = \omega + \phi_1 \sigma_{t-1}^2 + \kappa_1 y_{t-1}^2$ with $\sigma_\varepsilon^2 = 1$, see Francq and Zakoïan (2010) for a recent survey of GARCH models. If the financial return in question is predictable, then y_t can be interpreted as de-meaned return, i.e. the unpredictable part of return. However, more generally, y_t can be viewed as the error-term in a dynamic regression. Three characteristics that are often exhibited by financial returns are leverage (i.e. volatility asymmetry), conditional fat-tailedness and conditional skewness. The former means volatility tends to be higher after negative returns – this is typically attributed to leverage (hence the name), whereas conditional fat-tailedness means the standardised conditional return (i.e. ε_t) is more fat-tailed than the Gaussian. Conditional skewness means the standardised return is not symmetric. For stock returns, the skewness is typically negative, which means the probability of a large negative return is greater than a large positive return, even after controlling or adjusting for the recent level of volatility.

Several R packages provide facilities for the estimation and forecasting of univariate GARCH models that contains one or more of these features. Arguably, the three most important packages are **tseries** by Trapletti and Hornik (2013), **fGarch** by Wuertz et al. (2013) and **rugarch** by Ghalanos (2013). The **tseries** package probably has the fastest GARCH optimiser, but does not offer state-of-the-art specifications with leverage and fat-tailed skewed densities. This, by contrast, is provided by the **fGarch** and **rugarch** packages. The former has been considered by many — including myself — as the benchmark package in R for quite a while, since it provides a wide range of GARCH models coupled with a variety of densities. However, unfortunately, **fGarch** does not offer the possibility to estimate Exponential GARCH (EGARCH) models, i.e. models where the dynamics is specified in terms of $\ln \sigma_t^2$ rather than in terms of σ_t^2. EGARCH models are attractive, since they enable richer and more flexible dynamics (parameters can be negative), and since the autocorrelation function of returns depends on the conditional density (this is not the case for non-exponential ARCH models). The **rugarch** package, which despite its relative recent origin (available on CRAN since September 2011) already offers an impressive range of GARCH specifications and variations of these, fills this gap to some extent by providing Nelson's (1991) EGARCH model. Another package that partially fills this gap is **AutoSEARCH** by Sucarrat (2012), which offers estimation and automated General-to-Specific model selection of log-ARCH-X models. However, to the best of my knowledge, there are no other R packages that provide facilities for additional EGARCH models. The **betategarch** package thus contributes to the R world by offering utilities for the simulation, estimation and forecasting of Beta-t-EGARCH models.

The Beta-t-EGARCH model was first proposed by Harvey (2013) and Harvey and Chakravarty (2008), but it can also be viewed as an unrestricted version of the Generalised Autoregressive Score

(GAS) model of Creal et al. (2013). The code upon which **betategarch** is based was originally developed for Harvey and Sucarrat (2013), which extends the Beta-t-EGARCH model to the skewed case. The Beta-Skew-t-EGARCH model has a number of attractions. First, the model is robust to jumps or outliers, and fares very well empirically for a variety of financial returns when compared with other GARCH models, see Harvey and Sucarrat (2013). Second, the model accommodates the most important characteristics of time-varying financial volatility: Leverage, conditional fat-tailedness, conditional skewness and a decomposition of volatility into a short-term and a long-term component. Third, the unconditional moments of return (i.e. y_t) exist (if the conditional moments exist), which is important in long-term forecasting and for the computation of the autocorrelation function of returns. By contrast, this is generally not the case for Nelson's (1991) EGARCH when coupled with a t density: A necessary condition for the unconditional moments of the first-order specification to exist when ε_t is t is that the ARCH parameter is negative, see condition (A1.6) and the subsequent discussion in Nelson (1991, p. 365). Moreover, in the presence of leverage the ARCH parameter must be even more negative for the unconditional moments to exist. This is why Nelson (1991) proposed his model with a Generalised Error Distribution (GED) instead of a t. Fourth, the asymptotic properties are much easier to derive than those of Nelson's (1991) EGARCH, see Straumann and Mikosch (2006) and Wintenberger (2012) for a detailed description of the difficulties. Finally, since the conditional score drives the dynamics of the model, the Beta-t-EGARCH acquires some attractive theoretical properties. In particular, a simple transformation of the score is Beta-distributed (hence the name).

The two main functions of the **betategarch** package (version 3.1) are `tegarchSim` and `tegarch`. The first simulates from a Beta-Skew-t-EGARCH, whereas the latter estimates one. The `tegarch` function returns an object (a list) of the `tegarch` class, and a collection of S3 methods developed for this class can be applied to such objects: `coef`, `fitted`, `logLik`, `predict`, `print`, `residuals`, `summary` and `vcov`. The rest of the functions in the package are either auxiliary functions called by the two main functions, or a dataset, `nasdaq`, which is included for illustration purposes (see empirical example below). Finally, it is worth noting that the objects returned by `tegarchSim` function and the `fitted` and `predict` methods are of the class `zoo` defined in package **zoo**, see Zeileis et al. (2013). This means a large range of useful time-series methods and facilities can be applied to these objects, including plotting and printing methods.

The one-component Beta-Skew-t-EGARCH

The martingale difference version of the first order one-component Beta-Skew-t-EGARCH model (see Sections 4 and 6 in Harvey and Sucarrat (2013)) is given by

$$\begin{aligned} y_t &= \exp(\lambda_t)\varepsilon_t = \sigma_t\varepsilon_t, \qquad \varepsilon_t \sim st(0,\sigma_\varepsilon^2,\nu,\gamma), \qquad \nu > 2, \qquad \gamma \in (0,\infty), && (1)\\ \lambda_t &= \omega + \lambda_t^\dagger, && (2)\\ \lambda_t^\dagger &= \phi_1\lambda_{t-1}^\dagger + \kappa_1 u_{t-1} + \kappa^* sgn(-y_{t-1})(u_{t-1}+1), \qquad |\phi_1| < 1. && (3)\end{aligned}$$

The σ_t is the conditional scale or volatility, which need not equal the conditional standard deviation. In other words, ε_t is not standardised to have variance one. The conditional standard deviation is obtained as $\sigma_t\sigma_\varepsilon$, where σ_ε^2 is the variance of ε_t. The conditional error ε_t is distributed as a skewed t with zero mean, scale σ_ε^2, degrees of freedom parameter ν and skewness parameter γ. The conditional error is defined as $\varepsilon_t = \varepsilon_t^* - \mu_{\varepsilon^*}$, where ε_t^* is an uncentred (i.e. mean not necessarily equal to zero) Skewed t variable with ν degrees of freedom, skewness parameter γ and mean μ_{ε^*}. A centred and symmetric (i.e. ordinary) t-distributed variable with mean zero is obtained when $\gamma = 1$, in which $\mu_{\varepsilon^*} = 0$, whereas a left-skewed (right-skewed) t-variable is obtained when $\gamma < 1$ ($\gamma > 1$). More details on the distribution are given below. The ω is the log-scale intercept and can be interpreted as the long-term log-volatility, ϕ_1 is the persistence parameter (the bigger, the more clustering), κ_1 is the ARCH parameter (the bigger in absolute value, the greater the response to shocks), u_t is the conditional score (i.e. the derivative of the log-likelihood of y_t at t with respect to λ_t) and κ^* is the leverage parameter. A sufficient condition for stability in λ_t is $|\phi_1| < 1$.

Let ϵ^* denote an ordinary (i.e. centred and symmetric) t-distributed variable (with unit scale), and let $f(\epsilon^*)$ denote its density. By means of the skewing method of Fernández and Steel (1998), the density of an uncentred Skewed t variable can be written as

$$f(\varepsilon^*|\gamma) = \frac{2}{\gamma+\gamma^{-1}} f\left(\frac{\varepsilon^*}{\gamma^{sgn(\varepsilon^*)}}\right). \qquad (4)$$

Computation of the density values and of the mean of an uncentred Skewed t variable, and random number generation, can be undertaken with `dST`, `STmean` and `rST`, respectively. For example, the following code compares the empirical average of ten thousand random numbers with the analytical mean:

```
library(betategarch)
set.seed(123)
eps <- rST(10000, df=5, skew=0.7)
mean(eps)
[1] -0.69805
STmean(df=5, skew=0.7)
[1] -0.6914265
```

In addition, the functions `STvar`, `STskewness` and `STkurtosis` return the analytical values of the variance, skewness (the standardised 3rd moment) and kurtosis (the standardised 4th moment), respectively, of an uncentered Skewed t variable.

The conditional score of the martingale difference version of the Beta-Skew-t-EGARCH model (see Harvey and Sucarrat (2013, equation (32) in section 4)) is given by

$$\begin{aligned}\frac{\partial \ln f_y(y_t)}{\partial \lambda_t} &= u_t \\ &= \frac{(\nu+1)[y_t^2 + y_t \mu_{\varepsilon^*} \exp(\lambda_t)]}{\nu \exp(2\lambda_t)\gamma^{2sgn(y_t+\mu_{\varepsilon^*}\exp(\lambda_t))} + (y_t + \mu_{\varepsilon^*}\exp(\lambda_t))^2} - 1. \qquad (5)\end{aligned}$$

It is useful to note, however, that for simulation purposes the score u_t can also be written more conveniently as

$$u_t = \frac{(\nu+1)(\varepsilon_t^{*2} - \mu_{\varepsilon^*}\varepsilon_t^*)}{\nu\gamma^{2sgn(\varepsilon_t^*)} + \varepsilon_t^{*2}} - 1,$$

where ε_t^* is an uncentred Skewed t variable. When the conditional distribution is symmetric (i.e. $\gamma = 1$), then $\frac{u_t+1}{\nu+1} \sim Beta(1/2, \nu/2)$. This explains the origin of the name Beta-t-EGARCH.

Financial returns that follow the one-component Beta-Skew-t-EGARCH model given by (1)-(3) can be simulated with the `tegarchSim` function. For example, in order to generate two thousand returns from a specification with empirically plausible values on ω, ϕ_1, κ_1 and ν, but without leverage and skewness, then the following code can be used:

```
y1 <- tegarchSim(2000, omega=0.1, phi1=0.95, kappa1=0.05, df=10)
```

Similarly, the following three commands each generate two thousand returns with, respectively, moderate leverage, strong left-skewness, and both moderate leverage and strong left-skewness:

```
y2 <- tegarchSim(2000, omega=0.1, phi1=0.95, kappa1=0.05, df=10, kappastar=0.02)
y3 <- tegarchSim(2000, omega=0.1, phi1=0.95, kappa1=0.05, df=10, skew=0.8)
y4 <- tegarchSim(2000, omega=0.1, phi1=0.95, kappa1=0.05, df=10, kappastar=0.02,
       skew=0.8)
```

By default, the `tegarchSim` function returns the values of y_t only. However, for the full set of output one may use the `verbose` option. For example, the following code generates 100 observations using the default parameter values, stores the output in the matrix `mY` that is of class `zoo` and returns the first six observations:

```
mY <- tegarchSim(100, verbose=TRUE)
head(mY)

            y     sigma    stdev       lambda   lambdadagg          u     epsilon
1  0.19977534 1.0000000 1.118034  0.000000000  0.000000000 -0.9562733  0.19977534
2 -1.35118283 0.9904828 1.107393 -0.009562733 -0.009562733  0.7258681 -1.36416581
3  0.15475640 0.9981758 1.115994 -0.001825916 -0.001825916 -0.9736225  0.15503923
4 -0.04853563 0.9885947 1.105282 -0.011470845 -0.011470845 -0.9973492 -0.04909558
5  0.48034223 0.9793455 1.094942 -0.020870795 -0.020870795 -0.7415964  0.49047270
6  0.39742433 0.9731245 1.087986 -0.027243220 -0.027243220 -0.8195400  0.40840028
```

The last column named "epsilon" contains the centred Skewed t variable ε_t as defined in (1). The zeros for λ_t and $\lambda_t^\dagger$ in the first row are due to the default initial value. This can be changed via the `lambda.initial` option.

The two-component Beta-Skew-t-EGARCH

Squared financial return often exhibit long-memory, see Ding et al. (1993), Ding and Granger (1996). Two-component models of volatility accommodates the long-memory property by decomposing volatility into one long-term component and one short-term component. The role of the latter is to pick up temporary changes following a shock.

The martingale difference version of the first order two-component Beta-Skew-t-EGARCH model (see sections 2.5 and 6 in Harvey and Sucarrat (2013)) is given by

$$\begin{aligned}
y &= \exp(\lambda_t)\varepsilon_t = \sigma_t\varepsilon_t, \qquad \varepsilon_t \sim st(0,\sigma_\varepsilon^2,\nu,\gamma), \qquad \nu,\gamma \in (0,\infty), && (6)\\
\lambda_t &= \omega + \lambda_{1,t}^{\dagger} + \lambda_{2,t}^{\dagger}, && (7)\\
\lambda_{1,t}^{\dagger} &= \phi_1\lambda_{1,t-1}^{\dagger} + \kappa_1 u_{t-1}, \qquad |\phi_1| < 1, && (8)\\
\lambda_{2,t}^{\dagger} &= \phi_2\lambda_{2,t-1}^{\dagger} + \kappa_2 u_{t-1} + \kappa^* sgn(-y_{t-1})(u_{t-1}+1), \qquad |\phi_2| < 1, \qquad \phi_1 \neq \phi_2. && (9)
\end{aligned}$$

The $\lambda_{1,t}$ and $\lambda_{2,t}$ can be interpreted as the time-varying long-term and short-term components of log-volatility, respectively. The conditional score u_t, also here given by (5), drives both the long-run and short-run components, but leverage appears only in the short-term component. This is in line with the view that shocks only matters for short-term volatility, see e.g. Engle and Lee (1999). The model is not identifiable if $\phi_2 = \phi_1$.

Returns that follow a two-component Beta-Skew-t-EGARCH model given by (6)-(9) can also be simulated with the `tegarchSim` function. For example, the following code generates three thousand returns from a specification with empirically plausible values on the other parameters, but without leverage and skewness:

```
y1 <- tegarchSim(3000, omega=0.2, phi1=0.98, phi2=0.9, kappa1=0.01, kappa2=0.02, df=5)
```

Similarly, just as in the one-component case, the following code generates three thousand values of y_t with, respectively, leverage, skewness, and both leverage and skewness:

```
y2 <- tegarchSim(3000, omega=0.2, phi1=0.98, phi2=0.9, kappa1=0.01, kappa2=0.02,
        kappastar=0.04, df=5)
y3 <- tegarchSim(3000, omega=0.2, phi1=0.98, phi2=0.9, kappa1=0.01, kappa2=0.02,
        df=5, skew=0.95)
y4 <- tegarchSim(3000, omega=0.2, phi1=0.98, phi2=0.9, kappa1=0.01, kappa2=0.02,
        kappastar=0.04, df=5, skew=0.95)
```

Also here is the `verbose` option available for a more detailed output, and also here can the `lambda.initial` option be used to change the initial values.

Estimation and inference

One-component and two-component specifications can be estimated with the `tegarch` function. For example, the following code generates 5000 values of y_t with default parameter values, estimates a one-component specification with leverage and skewness, and then prints the most important information:

```
set.seed(123)
y <- tegarchSim(5000)
onecompmod <- tegarch(y)
onecompmod

Date: Wed Dec 04 19:53:52 2013
Message (nlminb): relative convergence (4)

Coefficients:
                  omega       phi1      kappa1    kappastar       df       skew
Estimate:   -0.002491487 0.92076991 0.014298775 -0.003284977 9.371817 0.99867519
Std. Error:  0.018374338 0.04263685 0.005027258  0.003574193 1.087466 0.01979645

Log-likelihood: -7635.482215
BIC:                3.064414
```

Estimation without leverage or skewness or both can be achieved by setting the `asym` and `skew` options to `FALSE`. For alternative summaries of the estimation results the `summary` method can be used, either with its `verbose` option set to `FALSE` (default) or `TRUE` (more information is returned). The latter returns, amongst other, the intial values used, the upper and lower bounds used and the numerically estimated Hessian. For additional inference on the parameters the covariance-matrix of the parameter estimates can be extracted with the `vcov` method:

```
vcov(onecompmod)

                  omega          phi1        kappa1     kappastar            df
omega      3.376163e-04  4.821495e-06 -3.369968e-06  3.781322e-06  1.202316e-02
phi1       4.821495e-06  1.817901e-03 -1.321280e-04  7.384223e-05  1.460351e-05
kappa1    -3.369968e-06 -1.321280e-04  2.527332e-05 -5.374307e-06 -5.498031e-04
kappastar  3.781322e-06  7.384223e-05 -5.374307e-06  1.277486e-05  2.261583e-05
df         1.202316e-02  1.460351e-05 -5.498031e-04  2.261583e-05  1.182581e+00
skew       1.808787e-06 -6.327049e-05  3.869190e-06 -7.473051e-06  2.075498e-04
                   skew
omega      1.808787e-06
phi1      -6.327049e-05
kappa1     3.869190e-06
kappastar -7.473051e-06
df         2.075498e-04
skew       3.918993e-04
```

In order to estimate a two-component Beta-Skew-t-EGARCH model the `components` option has to be set to 2:

```
twocompmod <- tegarch(y, components=2)
```

To estimate a two-component model without skewness the `skew` argument must be set to `FALSE`. Leverage, however, cannot be turned off in the two-component model for identifiability reasons.

Forecasting volatility

Forecasts of volatility — i.e. either the conditional standard deviation or the conditional scale — can be generated with the `predict` method applied to a `tegarch` object. The formula for n-step ahead forecasts of conditional scale σ_t is

$$E_t(\sigma_{t+n}) = \exp(\omega + \phi_1^n \lambda_t^\dagger) \cdot \Pi_{i=1}^n E_t \left[\exp(\phi_1^{n-i} g_{t+i-1})\right] \tag{10}$$

for the one-component model, where g_t is an IID term equal to $\kappa_1 u_t + \kappa^* sgn(-\varepsilon)(u_t + 1)$. The t subscript in the conditional expectation operator $E_t(\cdot)$ means the set of conditioning information contains all values up to and including period t. Accordingly, for $i = 1$ the value of $E_t[\exp(\phi_1^{n-i} g_{t+i-1})]$ is $\exp(\phi_1^{n-1} g_t)$. For $i > 1$, however, the expectations are not available in explicit form, so they are estimated by computing the sample mean of a large number of simulated IID variates. The default number of IID variates is 10 000, but this can be changed via the `n.sim` option. Another default option is that the `predict` method only returns forecasts of the conditional standard deviation

$$E_t(\sigma_{t+n})\sigma_\varepsilon. \tag{11}$$

However, if the `verbose` option is changed to `TRUE`, then forecasts of the conditional scale are also returned. Similarly, the initial values used are by default those of the last observation in the estimation sample. This can be altered via the `initial.values` option, e.g. for counterfactual or scenario analysis purposes.

The scale formula for the two-component specification is

$$E_t(\sigma_{t+n}) = \exp(\omega + \phi_1^n \lambda_{1,t}^\dagger + \phi_2^n \lambda_{2,t}^\dagger) \cdot \Pi_{i=1}^n E_t \left[\exp(\phi_1^{n-i} g_{1,t+i-1} + \phi_2^{n-i} g_{2,t+i-1})\right], \tag{12}$$

where $g_{1,t} = \kappa_1 u_t$ and $g_{2,t} = \kappa_2 u_t + \kappa^* sgn(-\varepsilon)(u_t + 1)$. Again, forecasts of conditional standard deviations are given by $E_t(\sigma_{t+n})\sigma_\varepsilon$, and the expectations in the last term on the right are estimated by simulation for $i > 1$.

As an example, the following code generates forecasts up to 7-period ahead for both scale and standard deviation for the one-component model estimated above:

```
set.seed(123)
predict(onecompmod, n.ahead=7, verbose=TRUE)

     sigma    stdev
1 1.012363 1.141463
2 1.014470 1.143838
3 1.012704 1.141847
```

```
4 1.011081 1.140017
5 1.009589 1.138335
6 1.008217 1.136788
7 1.006955 1.135365
```

The returned object is of class zoo, so a convenient time-series plotting method is readily available. Similarly, the command `predict(twocompmod,n.ahead=7,verbose=TRUE)` generates a corresponding set of forecasts for the two-component model estimated above.

Computational challenges

Estimation of the Beta-Skew-t-EGARCH model is by exact Maximum Log-likelihood (ML). The expressions of the first and second derivatives are not available in explicit form, so the procedure is all numerical. This leads to four computational challenges. The first is simply that the model is highly nonlinear, which is readily apparent by simply looking at the expression for the score (equation (5)). Moreover, the degree of non-linearity is compounded in the two-component specification. However, as no positivity constraints on the ARCH, GARCH and leverage parameters (i.e. $\omega, \phi_1, \phi_2, \kappa_1, \kappa_2, \kappa$) are needed, the numerical challenges are in fact not as large as those of, say, the two-component GARCH model of Engle and Lee (1999). There, positivity constraints on all parameters are necessary. As in all highly nonlinear estimation problems a set of good initial values is essential. By default, these are 0.02, 0.95, 0.05, 0.01, 10, 0.98 for one-component specifications, and 0.02, 0.95, 0.9, 0.001, 0.01, 0.005, 10, 0.98 for two-component specifications. However, if the user wishes to do so they can be changed via the `initial.values` option in the `tegarch` function. The `summary` method with option `verbose=TRUE` returns (amongst other) the initial values used in estimation.

The second computational challenge is that dynamic stability or stationarity — in practice that $|\phi_1| < 1$ (and $|\phi_2| < 1$ in the two-component case) — is required for estimation, whereas empirically ϕ_1 (and ϕ_2) is often close to, but just below, 1. In order to avoid explosive recursions during estimation it is therefore desirable to restrict ϕ_1 (and ϕ_2) such that $|\phi_1| < 1$ (and $|\phi_2| < 1$). My experience suggests the `nlminb` function is an excellent choice for this type of problems. The `optim` function with the L-BFGS-U option provides a similar bounding facility, but in practice it does not work as well as `nlminb` (it is less precise and fails more often in my experience). As for bounds, the skewing parameter γ is only defined on strictly positive values, and on theoretical grounds the degrees of freedom parameter ν must be greater than 2. For these reasons the default lower bounds on ϕ_1, ϕ_2, ν and γ are `-1+.Machine$double.eps`, `-1+.Machine$double.eps`, `2+.Machine$double.eps` and `.Machine$double.eps`, and their default upper bounds are `1-.Machine$double.eps`, `1-.Machine$double.eps`, `+Inf` and `+Inf` (i.e. ν and γ are unbounded from above). The other parameters are unbounded (i.e. their default upper and lower bounds are `+Inf` and `-Inf`, respectively). If the user wishes to do so the bounds can be changed via the `lower` and `upper` options in the `tegarch` function. Additional control options can also be passed on to the `nlminb` optimiser (see `nlminb` documentation) by introducing them as arguments in the `tegarch` function.

A third computational challenge is due to the presence of the sign function *sgn* in the skewed density (4), which means the log-likelihood is not differentiable in γ at the skewness change point. The log-likelihood is continuous, so the problem is numerical and not theoretical. In fact, consistency and asymptotic normality results often holds regardless (see e.g. Zhu and Galbraith (2010)). Most of the time the user will not encounter problems due to this characteristic, neither in simulation nor in empirical practice. Occasionally, however, the numerically estimated gradients (analytical ones are not available in explicit form) may explode when they iterate towards the proximity of the skewness change point. The `nlminb` function together with its bounding facility resolves this problem to a large extent. For extra protection against `NA` and `+/-Inf` values the log-likelihood functions (`tegarchLogl` and `tegarchLogl2`) check for `NA` and `+/-Inf` values at each iteration: Whenever such values are produced, then a small value on the log-likelihood is returned. By default this small value is the log-likelihood associated with the initial values, but this can be changed via the `logl.penalty` option in the `tegarch` function.

The fourth computational challenge is easily resolved. The expressions for the density function of the *t*-distribution and the moments of a *t*-distributed variable contain the `gamma` function in both the numerator and the denominator. When the argument of the `gamma` function is 172 or larger (i.e. 344 degrees of freedom or higher), then a value of `Inf/Inf = NaN` is returned, which can be detrimental to estimation. This issue is not particular to R but a consequence of how the Gamma function is implemented numerically in computer languages in general. Luckily, the issue can be easily resolved by noting that `beta(x,y) = gamma(x) * gamma(y)/gamma(x+y)`. The *t*-density and the moments of *t*-variables are thus readily re-written in terms of the `beta` function. The **fGarch** package has been using this parametrisation of the *t*-distribution for some time (always?), and I suspect it is for the same reason.

Empirical example: Daily volatility of the Nasdaq 100 stock market index

This section illustrates the use of the package in an empirical example (it is available as a demo by typing `demo(betategarch)`). The financial return in question is the daily log-return (in percent) of the Nasdaq 100 composite index (adjusted closing values). The study by Harvey and Sucarrat (2013) suggests stock market indices are particularly prone to be characterised by leverage, conditional fat-tailedness, skewness and a long-term component, so a stock market index is therefore specially suited to illustrate the usefulness of the Beta-Skew-t-EGARCH model. Also, the Nasdaq index was not included in Harvey and Sucarrat (2013) study. The source of the data is `http://finance.yahoo.com/` and the period in question is 3 January 2001 to 15 October 2013, i.e. a total of 3215 observations.

The following code loads the data, defines `y` to equal daily return in terms of a **zoo** object and plots `y`:

```
data(nasdaq)
y <- zoo(nasdaq[,"nasdaqret"], order.by=as.Date(nasdaq[,"day"], "%Y-%m-%d"))
plot(y, main="The Nasdaq 100 index (daily)", xlab="", ylab="Log-return in %")
```

The plot appears as the upper graph in Figure 1 and shows that the return series is clearly characterised by time-varying volatility.

To estimate a one-component Beta-Skew-t-EGARCH, to extract its fitted values and to plot the fitted conditional standard deviations, the following code can be used:

```
nasdaq1comp <- tegarch(y)
nasdaq1stdev <- fitted(nasdaq1comp)
plot(nasdaq1stdev, main="", ylab="1-comp: St.dev.", xlab="")
```

The estimation results are stored in `nasdaq1comp`, so typing `nasdaq1comp` yields

```
Date: Mon Dec 09 17:42:06 2013
Message (nlminb): relative convergence (4)

Coefficients:
                omega        phi1      kappa1   kappastar        df       skew
Estimate:   1.0421017 0.996543407 0.023508613 0.032033017 10.336337 0.85670426
Std. Error: 0.2412326 0.001184624 0.003542337 0.003065121  1.646172 0.01925872

Log-likelihood: -5586.666891
BIC:                3.490447
```

The degrees of freedom in the skewed t is estimated to be 10.3, a reasonably fat-tailed conditional t density, whereas the skewness is estimated to be about 0.86, which corresponds to pronounced negative skewness in ε_t. Also, the test statistic $(0.8567 - 1)/0.0193 = -7.4249$ is significant at all conventional significance levels. For a model without skewness, the command `tegarch(y,skew=FALSE)` can be used. For a closer look at the standardised residuals $\hat{\varepsilon}_t/\hat{\sigma}_\varepsilon$ in the estimated model above, the `residuals` method can be used for extraction. For the unstandardised residuals $\hat{\varepsilon}_t$, the `standardised` argument must be set to `FALSE`. `BIC` is the average value of the Schwarz (1978) information criterion. By default, the `fitted` method returns the fitted conditional standard deviation. However, more output is available by using the `verbose` option, i.e. by typing `fitted(nasdaq1comp,verbose=TRUE)`. This returns a matrix with, amongst other, the fitted scale and log-scale, the estimated score and the residuals. The returned matrix is a **zoo** object that is automatically matched with the dates — if any — of returns y_t. The middle graph in Figure 1 contains the plot of the fitted conditional standard deviations of the one-component model.

To estimate a two-component Beta-Skew-t-EGARCH model, to extract its fitted values and to plot its fitted conditional standard deviation, the following code can be used:

```
nasdaq2comp <- tegarch(y, components=2)
nasdaq2stdev <- fitted(nasdaq2comp)
plot(nasdaq2stdev, main="", ylab="2-comp: St.dev.", xlab="")
```

The plot of the fitted conditional standard deviations appears as the lower graph in Figure 1. Typing `nasdaq2comp` returns

```
Date: Mon Dec 09 17:42:09 2013
Message (nlminb): relative convergence (4)

Coefficients:
                omega          phi1       phi2      kappa1      kappa2   kappastar        df
```

```
Estimate:   1.4409972 1.0000000000 0.94175462 0.022074604 0.006442775 0.049203985 9.732886
Std. Error: 0.1991004 0.0004089023 0.01940377 0.004854267 0.006545395 0.006899511 1.564813
                  skew
Estimate:   0.89320223
Std. Error: 0.02094124

Log-likelihood: -5573.471563
BIC:                3.487262
```

Comparison of the BIC values of the two models suggest the latter provides a better fit. This provides further evidence in favour of two-component models in modelling the volatility of financial stock-market returns.

To generate out-of-sample forecasts up to 5-days ahead, the following code can be used:

```
set.seed(123)
predict(nasdaq1comp, n.ahead=5)

        1         2         3         4         5
0.8121401 0.8179406 0.8218067 0.8256776 0.8295533
```

In other words, the predicted conditional standard deviation 5 trading days after the 15th of October 2013 is about 0.8296. By default, the fitted values of λ_t and $\lambda_t^\dagger$ for the last day of the estimation sample are used as initial values. However, for alternative scenario and counterfactual analysis can be undertaken by providing user-specific values on λ_t and $\lambda_t^\dagger$ via the `initial.values` option.

Conceptually the Beta-Skew-t-EGARCH model can appear complicated. So one may ask whether it performs better than conceptually simpler models like, say, an ordinary GARCH model with leverage and the two-component model of Engle and Lee (1999). An ordinary GARCH(1,1) model with leverage of the Glosten et al. (1993) type, sometimes referred to as the GJR-GARCH, coupled with a standardised skewed t conditional density is given by

$$\begin{aligned} y &= \sigma_t \varepsilon_t, \qquad \varepsilon_t \sim st(0,1,\nu,\gamma), &(13)\\ \sigma_t^2 &= \omega + \phi_1 \sigma_{t-1}^2 + \kappa_1 y_{t-1}^2 + \kappa^* I_{\{y_{t-1}<0\}} y_{t-1}^2, &(14)\end{aligned}$$

where the parameters have similar interpretations to those of the Beta-Skew-t-EGARCH specification. The model can be estimated with the **fGarch** package by means of

```
library(fGarch)
nasdaqgarch <- garchFit(data=y, cond.dist="sstd",
  include.mean=FALSE, include.skew=TRUE, leverage=TRUE)
```

Next, `summary(nasdaqgarch)` yields (amongst other)

```
        Estimate  Std. Error  t value Pr(>|t|)
omega   0.016866    0.004315    3.908 9.29e-05 ***
alpha1  0.040237    0.009882    4.072 4.67e-05 ***
gamma1  0.712143    0.183734    3.876 0.000106 ***
beta1   0.933986    0.008357  111.762  < 2e-16 ***
skew    0.898964    0.021099   42.608  < 2e-16 ***
shape   9.910543    1.562340    6.343 2.25e-10 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Log Likelihood:
 -5591.593    normalized:  -1.73922

Information Criterion Statistics:
     AIC      BIC      SIC     HQIC
3.482173 3.493511 3.482166 3.486237
```

In the table `alpha1`, `gamma1` and `beta1` correspond to κ_1, κ^* and ϕ_1, respectively. The skewness and degrees of freedom estimates are relatively similar to those of the Beta-Skew-t-EGARCH above. However, the BIC value is lower, which means the Beta-Skew-t-EGARCH provides a better fit according to this criterion.

In the **rugarch** package the first order two-component model of Engle and Lee (1999) coupled with a standardised skewed t conditional density is given by

$$\begin{aligned} y &= \sigma_t \varepsilon_t, \qquad \varepsilon_t \sim st(0,1,\nu,\gamma), && (15) \\ \sigma_t^2 &= q_t + \phi_1(\sigma_{t-1}^2 - q_{t-1}) + \kappa_1(y_{t-1}^2 - \sigma_{t-1}^2), && (16) \\ q_t^2 &= \omega + \phi_2 q_{t-1}^2 + \kappa_1(y_{t-1}^2 - \sigma_{t-1}^2). && (17) \end{aligned}$$

In other words, **rugarch** does not provide the option to include leverage in the Engle and Lee (1999) model. Estimation can be undertaken with

```
library(rugarch)
EngleLeeSpec <- ugarchspec(variance.model = list(model = "csGARCH",
        garchOrder = c(1, 1)), mean.model=list(armaOrder=c(0,0),
        include.mean=FALSE), distribution.model="sstd")
nasdaqEngleLee <- ugarchfit(EngleLeeSpec, y, solver="nlminb")
```

Next, `summary(nasdaqgarch)` yields (amongst other)

```
        Estimate  Std. Error  t value Pr(>|t|)
omega   0.008419    0.003911   2.1528 0.031337
alpha1  0.021006    0.014765   1.4228 0.154807
beta1   0.931799    0.044489  20.9446 0.000000
eta11   0.995956    0.002383 417.9196 0.000000
eta21   0.044147    0.013570   3.2533 0.001141
skew    0.912973    0.020984  43.5076 0.000000
shape  11.055129    2.095500   5.2757 0.000000

LogLikelihood : -5617.186

Information Criteria
------------------------------------

Akaike       3.4987
Bayes        3.5119
Shibata      3.4987
Hannan-Quinn 3.5035
```

In the table `alpha1`, `beta1`, `eta11` and `eta21` correspond to κ_1, ϕ_1, ϕ_2 and κ_2, respectively. The skewness and degrees of freedom estimates are again relatively similar to those of the Beta-Skew-t-EGARCH above, and again the BIC value is higher. So also in this case does the Beta-Skew-t-EGARCH provide a better fit according to BIC. In fact, the log-likelihood of the Engle and Lee (1999) model is even lower than that of the GJR-GARCH, which contains fewer parameters. This suggests leverage, which is omitted from the **rugarch** implementation of the Engle and Lee (1999) model, is an important determinant of Nasdaq 100 return volatility.

Summary

This paper illustrates how the **betategarch** package can be used for the simulation, estimation and forecasting of one-component and two-component first order Beta-Skew-t-EGARCH models. The model allows for skewed and heavy-tailed t-distributed conditional errors, and leverage and a time-varying long-term component in the volatility specification. The two main functions of the package are `tegarchSim` and `tegarch`. The first simulates from a Beta-Skew-t-EGARCH model, either a one-component or two-component specification, whereas the latter function estimates one. The object (a list) returned by the second function is of class `tegarch`, and a collection of S3 methods can be applied to objects from this class: `coef`, `fitted`, `logLik`, `predict`, `print`, `residuals`, `summary` and `vcov`. Finally, the empirical illustration on daily Nasdaq 100 returns provides further in-sample evidence in favour of the Beta-Skew-t-EGARCH model.

Bibliography

T. Bollerslev. Generalized autoregressive conditional heteroscedasticity. *Journal of Econometrics*, 31: 307–327, 1986. [p138]

D. Creal, S. J. Koopmans, and A. Lucas. Generalized Autoregressive Score Models with Applications. *Journal of Applied Econometrics*, 28:777–795, 2013. [p139]

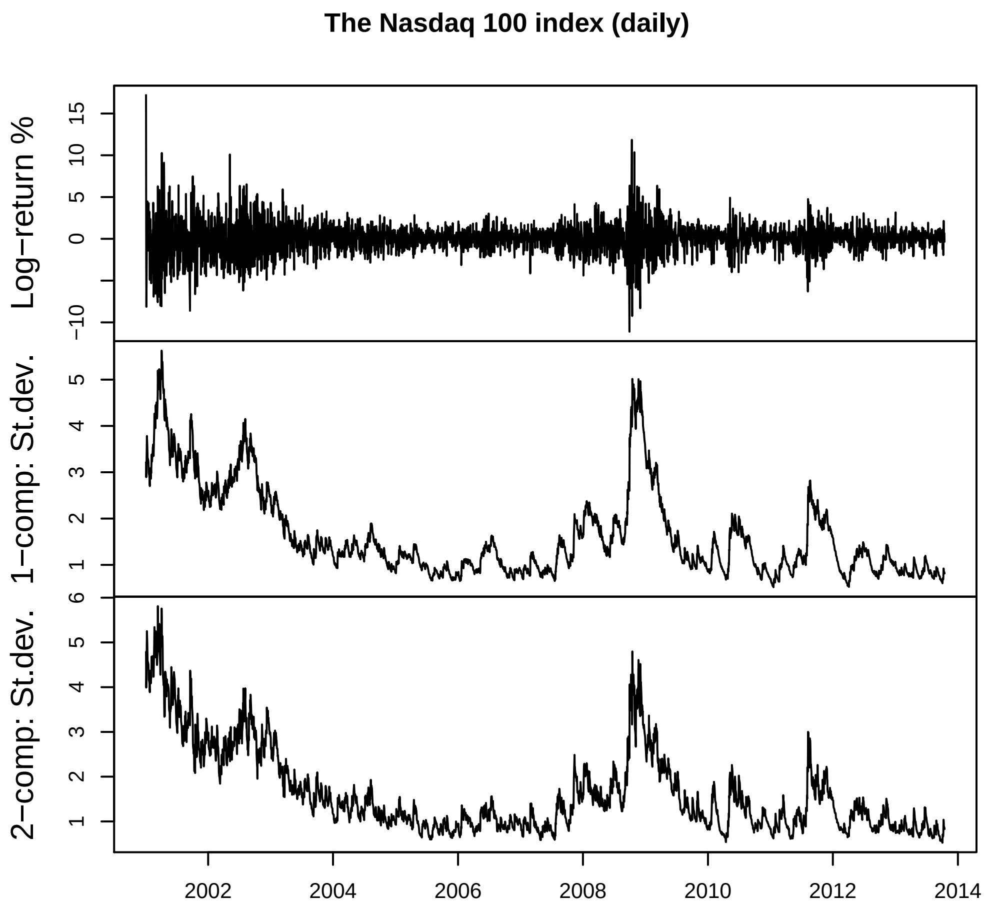


Figure 1: Nasdaq log-returns in %, fitted conditional standard deviation of the one-component model and fitted conditional standard deviation of the two-component model.

Z. Ding and C. Granger. Modelling volatility persistence of speculative returns: a new approach. *Journal of Empirical Finance*, 1:83–106, 1996. [p140]

Z. Ding, R. Engle, and C. Granger. A long memory property of stock market returns and a new model. *Journal of Empirical Finance*, 1:83–106, 1993. [p140]

R. Engle. Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflations. *Econometrica*, 50:987–1008, 1982. [p138]

R. F. Engle and G. G. Lee. A Long-Run and a Short-Run Component Model of Stock Return Volatility. Oxford University Press, Oxford, 1999. [p141, 143, 145, 146]

C. Fernández and M. Steel. On Bayesian Modelling of Fat Tails and Skewness. *Journal of the American Statistical Association*, 93:359–371, 1998. [p139]

C. Francq and J.-M. Zakoïan. *GARCH Models*. Marcel Dekker, New York, 2010. [p138]

A. Ghalanos. *rugarch: Univariate GARCH models*, 2013. URL `http://CRAN.R-project.org/package=rugarch`. R package version 1.2-7. [p138]

L. R. Glosten, R. Jagannathan, and D. E. Runkle. On the Relation between the Expected Value and the Volatility of the Nominal Excess Return on Stocks. *Journal of Finance*, 48:1779–1801, 1993. [p145]

A. C. Harvey. *Dynamic Models for Volatility and Heavy Tails*. Cambridge University Press, New York, 2013. [p138]

A. C. Harvey and T. Chakravarty. Beta-t-(E)GARCH. Cambridge Working Papers in Economics 0840, Faculty of Economics, University of Cambridge, 2008. [p138]

A. C. Harvey and G. Sucarrat. EGARCH models with fat tails, skewness and leverage. *Computational Statistics and Data Analysis*, Forthcoming, 2013. URL http://dx.doi.org/10.1016/j.csda.2013.09.022. [p139, 140, 141, 144]

D. B. Nelson. Conditional Heteroskedasticity in Asset Returns: A New Approach. *Econometrica*, 59: 347–370, 1991. [p138, 139]

G. Schwarz. Estimating the Dimension of a Model. *The Annals of Statistics*, 6:461–464, 1978. [p144]

D. Straumann and T. Mikosch. Quasi-Maximum-Likelihood Estimation in Conditionally Heteroscedastic Time Series: A Stochastic Recurrence Equations Approach. *The Annals of Statistics*, 34:2449–2495, 2006. [p139]

G. Sucarrat. *AutoSEARCH: General-to-Specific (GETS) Model Selection*, 2012. URL http://CRAN.R-project.org/package=AutoSEARCH. R package version 1.2. [p138]

A. Trapletti and K. Hornik. *tseries: Time series analysis and computational finance*, 2013. URL http://CRAN.R-project.org/package=tseries. R package version 0.10-32. [p138]

O. Wintenberger. QML estimation of the continuously invertible EGARCH(1,1) model. Unpublished working paper, 2012. [p139]

D. Wuertz, Y. C. with contribution from Michal Miklovic, C. Boudt, P. Chausse, and others. *fGarch: Rmetrics - Autoregressive Conditional Heteroskedastic Modelling*, 2013. URL http://CRAN.R-project.org/package=fGarch. R package version 3010.82. [p138]

A. Zeileis, G. Grothendieck, and J. A. Ryan. *zoo: S3 Infrastructure for Regular and Irregular Time Series (Z's ordered observations)*, 2013. URL http://CRAN.R-project.org/package=zoo. R package version 1.7-10. [p139]

D. Zhu and J. W. Galbraith. A Generalized Asymmetric Student t Distribution with Application to Financial Econometrics. *Journal of Econometrics*, 157:297–305, 2010. [p143]

Genaro Sucarrat
BI Norwegian Business School
Nydalsveien 37
0484 Oslo
Norway
genaro.sucarrat@bi.no

Changes to grid for R 3.0.0

by Paul Murrell

Abstract From R 3.0.0, there is a new recommended way to develop new grob classes in **grid**. In a nutshell, two new "hook" functions, `makeContext()` and `makeContent()` have been added to **grid** to provide an alternative to the existing hook functions `preDrawDetails()`, `drawDetails()`, and `postDrawDetails()`. There is also a new function called `grid.force()`. This article discusses why these changes have been made, provides a simple demonstration of the use of the new functions, and discusses some of the implications for packages that build on **grid**.

Introduction

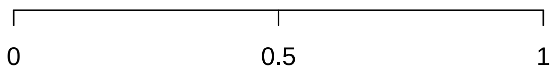
The **grid** graphics package (Murrell, 2011) provides a low-level graphics system as an alternative to the default **graphics** package. Several high-level graphics packages build on **grid**; for example, if we use **lattice** (Sarkar, 2008) or **ggplot2** (Wickham, 2009) to draw a plot, then we are also using **grid**.

This section shows a simple example of using **grid** that results in a problem, and this problem provides the motivation for the changes that were made to **grid** for R 3.0.0.

The following code uses the **grid** package to draw an axis.

```
> library(grid)
```

```
> grid.xaxis(at=c(0, .5, 1), name="axis-1")
```



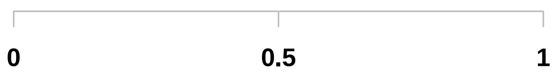
In addition to drawing the axis, **grid** keeps a list of graphical objects, or *grobs*, that contain descriptions of what has been drawn. The following code lists the grobs in the current scene: there is a parent grob called `"axis-1"` (this is actually a *gTree*, which is a grob that can have other grobs as children), and several child grobs including a "major" line, several "ticks" line segments, and several text "labels", all collected together to make an axis.

```
> grid.ls(fullNames=TRUE)
```

```
xaxis[axis-1]
  lines[major]
  segments[ticks]
  text[labels]
```

The **grid** package keeps a list of grobs because it can be useful to access, query, and modify the grobs in a scene. For example, the following code uses the `grid.edit()` function to change the lines on the axis to grey and the text to bold.

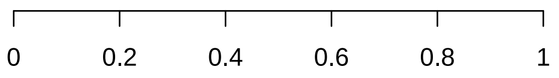
```
> grid.edit("major|ticks", grep=TRUE, global=TRUE, gp=gpar(col="grey"))
> grid.edit("labels", gp=gpar(fontface="bold"))
```



A problem

The next piece of code also draws an axis, but this time we do not specify where the tick marks should go on the axis. There is also code to show the grobs that **grid** has kept as a record of this scene. The important difference to note is that this time the grob listing only shows the `"axis-2"` gTree; there are no grobs representing the lines, segments, and text on the axis.

```
> grid.xaxis(name="axis-2")
```



```
> grid.ls(fullNames=TRUE)

xaxis[axis-2]
```

This lack of child grobs is a problem because it means that it is not possible to access (or query or modify) the child grobs. The problem exists because, when the tick mark locations are not specified for an axis, the axis decides which tick marks to draw every time the axis is drawn - no child grobs are kept because they are recreated every time.

One of the reasons for the changes to **grid** in R 3.0.0 is to provide a solution for this problem. It is important to point out that many **grid** grobs do not suffer from this issue at all. This problem only occurs for a small set of **grid** grobs that decide what to draw at drawing time rather than at creation time. On the other hand, the problem becomes more likely in packages that build on **grid** and define new classes of **grid** grobs, so fixing the problem in **grid** has large flow-on effects to other packages.

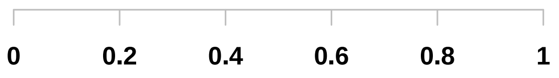
One of the very visible changes to **grid** is the new function `grid.force()`. The following code shows that the `grid.force()` function can be used to create permanent versions of the child grobs for the axis, which then means that it is possible to modify those child grobs.

```
> grid.force()

> grid.ls(fullNames=TRUE)

forcedgrob[axis-2]
  lines[major]
  segments[ticks]
  text[labels]

> grid.edit("major|ticks", grep=TRUE, global=TRUE, gp=gpar(col="grey"))
> grid.edit("labels", gp=gpar(fontface="bold"))
```



The `grid.force()` function is just one of the changes to **grid** for R 3.0.0. This article describes the full set of changes, including more about `grid.force()`, and explores some of the other reasons for change and some of the other benefits that arise from these changes.

A simple grid demonstration

In order to demonstrate the changes in **grid**, we will consider several different ways to develop a function that draws a "text box" with **grid**. This function will draw a text label and surround the text with a box (with rounded corners). In effect, we are going to create a new class of grob; one that draws text surrounded by a box.

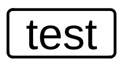
The simplest way to implement this sort of thing in **grid** is to write a function that makes several calls to draw standard **grid** grobs. For example, the following code defines a `textbox()` function that takes a single argument, a text label, and calls `textGrob()` to create a text grob, `roundrectGrob()` to create a box around the label, and then `grid.draw()` to draw the two grobs. The `stringWidth()` and `stringHeight()` functions are used to make sure that the box is the right size for the label.

```
> library(grid)

> textbox <- function(label) {
+     tg <- textGrob(label, name="text")
+     rr <- roundrectGrob(width=1.5*stringWidth(label),
+                         height=1.5*stringHeight(label),
+                         name="box")
+     grid.draw(tg)
+     grid.draw(rr)
+ }
```

The following code shows the function in action and the output is shown below the code.

```
> grid.newpage()
> textbox("test")
```



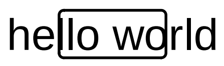
The code and output below shows that **grid** has kept a record of the grobs that were drawn.

```
> grid.ls(fullNames=TRUE)
```

```
text[text]
roundrect[box]
```

One deficiency with the `textbox()` function is that there is no connection between the two grobs that it creates. For example, if we modify the text (code below), the roundrect stays the same size and becomes too small for the text (see the output below the code).

```
> grid.edit("text", label="hello world")
```

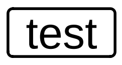


An alternative implementation is to *group* the two grobs together by constructing a gTree to contain them both. For example, the following code redefines the `textbox()` function so that it generates a gTree containing a text grob and a roundrect grob and then draws the gTree.

```
> textbox <- function(label) {
+     tg <- textGrob(label, name="text")
+     rr <- roundrectGrob(width=1.5*stringWidth(label),
+                         height=1.5*stringHeight(label),
+                         name="box")
+     gt <- gTree(children=gList(tg, rr), name="tb")
+     grid.draw(gt)
+ }
```

This version of the function produces the same output as the previous version, but the scene now consists of a single gTree that contains the text grob and the roundrect grob.

```
> grid.newpage()
> textbox("test")
```

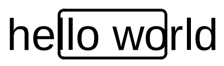


```
> grid.ls(fullNames=TRUE)
```

```
gTree[tb]
  text[text]
  roundrect[box]
```

Unfortunately, the contents of the gTree are fixed at creation time, so if we modify the text grob child of the gTree, the roundrect child is still not updated.

```
> grid.edit("tb::text", label="hello world")
```



The old drawDetails() hook

The behaviour of the text box can be made more coherent if we delay the construction of the box until drawing time (i.e., recalculate the box every time that we draw the text box). That way, the box will always be the right size for the text. This can be achieved in **grid** by creating a new class of grob with the `grob()` function and then defining a `drawDetails()` method for this new grob class. For all grobs, the `drawDetails()` hook is called whenever the grob is drawn (with the default `drawDetails()` method doing nothing).

For example, the following code redefines the `textbox()` function so that it generates a grob with the class `"textbox"` and draws that.

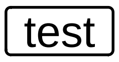
```
> textbox <- function(label,
+                     name=NULL, gp=NULL, vp=NULL) {
+     g <- grob(label=label,
+               name=name, gp=gp, vp=vp,
+               cl="textbox")
+     grid.draw(g)
+ }
```

Because we have created a new class of grob, **grid** does not know how to draw it. To tell **grid** how to draw a `"textbox"` grob, we can define a `drawDetails()` method for `"textbox"` grobs. Such a method is shown in the code below, which is almost identical to the previous version of `textbox()`; all that we have done is delay the generation of the text grob and roundrect grob until drawing time.

```
> drawDetails.textbox <- function(x, ...) {
+     tg <- textGrob(x$label, name="text")
+     rr <- roundrectGrob(width=1.5*stringWidth(x$label),
+                         height=1.5*stringHeight(x$label),
+                         name="box")
+     gt <- gTree(children=gList(tg, rr), name=x$name)
+     grid.draw(gt)
+ }
```

The following code shows the new `textbox()` function in action and shows that it produces exactly the same output as the first version.

```
> grid.newpage()
> textbox("test", name="tb")
```



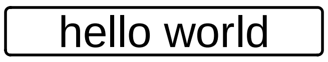
One big difference is that only one `"textbox"` grob was generated, rather than separate text and roundrect grobs. The latter are only generated at drawing time and are not retained.

```
> grid.ls(fullNames=TRUE)
```

```
textbox[tb]
```

The advantage that we get is that, if we modify that one grob, both the text and the box are updated.

```
> grid.edit("tb", label="hello world")
```



The disadvantage is that the individual text and box grobs are no longer visible as separate grobs, so it is not possible to access the individual text or roundrect grobs. In other words, we have a convenient high-level interface to the combined text and box, but we *only* have that high-level interface. This is the same problem that we had with the axis grob at the start of this article.

The new `makeContent()` hook

The new `makeContent()` function provides an alternative way to specify how to draw a new **grid** grob class (an alternative to writing a `drawDetails()` method). The main difference is that, whereas a `drawDetails()` method typically calls **grid** functions to *draw output*, a `makeContent()` method calls **grid** functions to *generate grobs*. The standard behaviour for grobs automatically takes care of drawing the content.

To continue our example, the following code redefines `textbox()` yet again. This is very similar to the previous version of `textbox()`. The one important difference in this new version is that the `gTree()` function is used to generate a new *gTree* class, rather than calling the `grob()` function to generate a new grob class. We do this because we are going to be writing a `makeContent()` method that creates more than one grob to draw; we can only use a `makeContent()` method for a grob class if the method only creates a single predefined grob to draw.[1] The gTree does not get any children when it is created because the children will be built (and added) at drawing time by a `makeContent()` method.

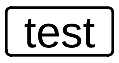
```
> textbox <- function(label,
+                     name=NULL, gp=NULL, vp=NULL) {
+     gt <- gTree(label=label,
+                 name=name, gp=gp, vp=vp,
+                 cl="textboxtree")
+     grid.draw(gt)
+ }
```

To tell **grid** how to draw this new gTree class, instead of a `drawDetails()` method, we define a `makeContent()` method. This is similar to the `drawDetails()` method above because it generates a text grob and a roundrect grob, but instead of drawing them, it simply adds these grobs as children of the gTree. The modified gTree must be returned as the result of this function so that **grid** can draw the generated content.

```
> makeContent.textboxtree <- function(x) {
+     t <- textGrob(x$label,
+                   name="text")
+     rr <- roundrectGrob(width=1.5*grobWidth(t),
+                         height=1.5*grobHeight(t),
+                         name="box")
+     setChildren(x, gList(t, rr))
+ }
```

The following code shows that the new `textbox()` function produces exactly the same output as before.

```
> grid.newpage()
> textbox("test", name="tbt")
```



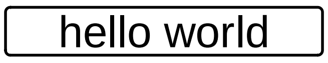
As with the `drawDetails()` approach, the scene consists of only one grob, this time a `"textboxtree"` grob.

```
> grid.ls(fullNames=TRUE)

textboxtree[tbt]
```

Furthermore, if we modify that one grob, both the text and the box are updated.

```
> grid.edit("tbt", label="hello world")
```



[1]We also create a different class than before, called `"textboxtree"` so that we do not have both `drawDetails()` and `makeContent()` methods defined for the same class.

In summary, the `makeContent()` approach behaves exactly the same as the `drawDetails()` approach. The advantages of the `makeContent()` approach lie in the *extra* things that it allows us to do.

The `grid.force()` function

The new function `grid.force()` affects any grobs that have a `makeContent()` method. This function *replaces* the original grob with the modified grob that is returned by the `makeContent()` method.

For example, if we use `grid.force()` on a scene that contains a `"textboxtree"` grob, the output of the scene is unaffected (see below).

```
> grid.force()
```

hello world

However, the scene now consists of a gTree with a text grob and a roundrect grob as its children (rather than just a single `"textboxtree"` object).

```
> grid.ls(fullNames=TRUE)
```

```
forcedgrob[tbt]
  text[text]
  forcedgrob[box]
```

Now that we can see the individual components of the text box, we can modify them independently. For example, the following code just modifies the box component of the scene, but not the text component.

```
> grid.edit("box", gp=gpar(col="grey"))
```

hello world

In other words, in addition to the convenient high-level interface to the text box, we can now "force" the high-level gTree to produce a low-level interface to the individual components of the text box.

Forced grobs

In the list of grobs above, the `"tbt"` grob is labelled as a `"forcedgrob"` after the call to `grid.force()`. This is an additional class that is attached to grobs that have been forced. The `"tbt"` grob is still a `"textboxtree"`, as shown below.

```
> class(grid.get("tbt"))
```

```
[1] "forcedgrob"  "textboxtree" "gTree"       "grob"        "gDesc"
```

The `"box"` grob in the example above has also been forced because **grid** `"roundrect"` grobs now have a `makeContent()` method. In this case, the forced grob is now a `"polygon"` grob (because the `makeContent()` method creates a `"polygon"` to draw based on the description in the `"roundrect"` grob).

```
> class(grid.get("box"))
```

```
[1] "forcedgrob" "polygon"    "grob"       "gDesc"
```

The `grid.revert()` function

One downside of calling `grid.force()` is that the convenient high-level interface to a grob is no longer available. For example, changing the label on the text box no longer has any effect.

```
> grid.edit("tbt", label="test")
```

hello world

The new `grid.revert()` function is provided to reverse the effect of `grid.force()` and replace the individual components of the forced grob with the original grob. The following code and shows this function in action. It also demonstrates that the reversion will lose any changes that were made to any of the individual components. We return to the scene that we had before the call to `grid.force()`.

```
> grid.revert()
```

hello world

In other words, for grobs that generate content at drawing time, we can have *either* the high-level interface *or* the low-level interface, but not both at once.

A reminder

All of the discussion in this article applies to the situation where a new **grid** grob class is created that *needs to calculate what to draw at drawing time*. If the entire content of a grob or gTree can be generated at creation time, rather than having to wait until drawing time, then things are much easier, and it is possible to have both a high-level interface and low-level access at the same time.

It is only when the content must be generated at drawing time, as is the case for **grid** axis grobs, that the design decisions and functions described in this article become necessary.

Review

To review the changes described so far, where once we might have written a `drawDetails()` method for a new **grid** grob or gTree class, we can *instead* write a `makeContent()` method. If we do so, the new `grid.force()` function can be used to gain access to low-level grobs that otherwise would not be accessible. One example where this is useful is for **grid** axis grobs (with no tick location specified), in order to gain access to the individual lines and text that make up the axis.

Revisiting the simple grid demonstration

In order to demonstrate some of the other changes in **grid** for R 3.0.0, we will revisit the simple text box example from before. In the implementations of the `textbox()` function so far, we have focused our effort on what *content* to draw for the text box. In this section, we also consider the *context* for drawing; the **grid** viewports that a text box is drawn within.

In this next implementation, when a text box is drawn, we will set up a viewport to draw the text box within and then draw a text grob and roundrect grob within that viewport. This will simplify the creation of the text and roundrect grobs.

This change only requires modifications to the methods for the `"textboxtree"` class; the `textbox()` function remains the same as before.

```
> textbox <- function(label,
+                     name=NULL, gp=NULL, vp=NULL) {
+     gt <- gTree(label=label,
+                 name=name, gp=gp, vp=vp,
+                 cl="textboxtree")
+     grid.draw(gt)
+ }
```

The old `preDrawDetails()` hook

We can specify how to set up the drawing context for a grob class by defining a `preDrawDetails()` method for the class. For all grobs, the `preDrawDetails()` hook is called *before* the `makeContent()` or `drawDetails()` hooks (with the default `preDrawDetails()` method doing nothing). The following code defines a method for `"textboxtree"` grobs that pushes a viewport the appropriate size for drawing the text box.

```
> preDrawDetails.textboxtree <- function(x) {
+     tbvp <- viewport(width=1.5*stringWidth(x$label),
+                      height=1.5*stringHeight(x$label))
+     pushViewport(tbvp)
+ }
```

With this method defined, the `makeContent()` method for `"textboxtree"` grobs becomes much simpler because the roundrect grob just fills up the viewport that was created by the `preDrawDetails()` method (we have already calculated the appropriate size when we created the viewport in the `preDrawDetails()` method).

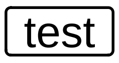
```
> makeContent.textboxtree <- function(x) {
+     t <- textGrob(x$label, name="text")
+     rr <- roundrectGrob(name="box")
+     setChildren(x, gList(t, rr))
+ }
```

Whenever a `preDrawDetails()` method is defined, it must be accompanied by a `postDrawDetails()` method, which must revert any changes to the drawing context.

```
> postDrawDetails.textboxtree <- function(x) {
+     popViewport()
+ }
```

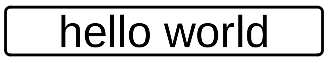
The following code shows that the `textbox()` function produces exactly the same output as before.

```
> grid.newpage()
> textbox("test", name="tbt")
```



The drawing context is regenerated every time the text box is drawn, so modifying the text label updates the viewport that both text and box are drawn in and the box expands with the text.

```
> grid.edit("tbt", label="hello world")
```



The new `makeContext()` hook

In parallel with the change from `drawDetails()` to `makeContent()`, there is a new `makeContext()` generic function to replace the use of `preDrawDetails()` (and `postDrawDetails()`).

The main difference is that a `makeContext()` method must create new viewports and *add* them to the vp slot of the grob (rather than pushing the new viewports), *and* it must return the modified grob. The following code demonstrates what a `makeContext()` method looks like for a `"textboxtree"` grob.

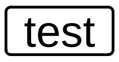
```
> makeContext.textboxtree <- function(x) {
+     tbvp <- viewport(width=1.5*stringWidth(x$label),
+                      height=1.5*stringHeight(x$label))
+     if (is.null(x$vp))
+         x$vp <- tbvp
+     else
+         x$vp <- vpStack(x$vp, tbvp)
+     x
+ }
```

This is similar to the `preDrawDetails()` method, but it has additional code to combine the new viewport with the current value of the `vp` slot for the grob. On the plus side, there is no need for a `postDrawDetails()` method. In fact, it is essential that we remove the `preDrawDetails()` and `postDrawDetails()` methods for this class; we only need the `makeContext()` method now.

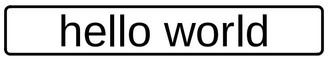
```
> rm("preDrawDetails.textboxtree")
> rm("postDrawDetails.textboxtree")
```

The following code shows that the `textbox()` function still works and that the box expands if we modify the text label.

```
> grid.newpage()
> textbox("test", name="tbt")
```



```
> grid.edit("tbt", label="hello world")
```



Mixing viewports with viewport paths

The example in the previous section contains another subtle change in **grid** for R 3.0.0. Within the `makeContext()` method there is the expression `vpStack(x$vp,tbvp)`. The `vpStack()` function combines two viewports into a *viewport stack* (one or more viewports that will be pushed in series, one after the other). The second argument to the call, `tbvp` is a viewport, but the first argument to the call is the `vp` slot of a grob, which could be a viewport *or* it could be a viewport path. The ability to combine viewport paths with viewports like this is new in R 3.0.0 and is necessary for `makeContext()` methods to work.

Another reminder

Modifying the drawing context at drawing time is not always necessary. When creating a new grob class, it is often simpler just to set up the drawing context at creation time by creating `childrenvp` for the children of a gTree. It is only when the generation of drawing context has to be delayed until drawing time that a `makeContext()` method becomes necessary.

Review

In addition to the new `makeContent()` hook for generating content at drawing time, and the new `grid.force()` function for exposing content that is only created at drawing time, there is a new `makeContext()` hook for generating context at drawing time.

The new `makeContext()` hook has been described, but it may not be clear what benefits accrue from it. That is the purpose of the next section.

Who benefits?

The problem with **grid** axes that was described at the start of this article has existed for many years. Having a solution for that problem scratches a years-long itch, but it was not the main reason for the changes in R 3.0.0. One of the motivations for changes to **grid** was provided by the difficulties that the authors of the **gtable** package (Wickham, 2012) had in implementing `"gtable"` grobs. The **gtable** package is important because it is used by the popular **ggplot2** package to arrange the different components of plots.

The **gtable** authors had to write not just `preDrawDetails()` and `postDrawDetails()` methods, but also `grid.draw()` methods to get the behaviour they desired for `"gtable"` grobs. This is undesirable because having a `grid.draw()` method makes `"gtable"` grobs behave differently from standard **grid** grobs. A symptom of this problem is the fact that not all grobs from a `"gtable"` are accessible for editing (similar to the axis problem at the start of this article). Having a special `grid.draw()` method also causes problems for packages that rely on the behaviour of standard **grid** grobs (as we shall see with the **gridSVG** package below).

The new `makeContext()` hook makes it possible to implement `"gtable"` grobs without resorting to a `grid.draw()` method. A new implementation is available as a fork of the **gtable** package on github.[2]

Another motivator for change was the **gridSVG** package (Murrell and Potter, 2013). The main function of this package is to transform every grob in a **grid** scene into an SVG representation, but this package could not transform grobs that were not accessible (such as the ticks and labels for an axis grob). In fact, the package could not transform any grob that had a `drawDetails()` method because the grobs produced by a `drawDetails()` method were not recorded anywhere. The new `grid.force()` function (which depends on the new `makeContent()` hook) means that the **gridSVG** package can now access all of the grobs in a scene by "forcing" the scene before transforming it.

A number of grobs within **grid** itself, such as `"roundrect"` grobs and `"curve"` grobs, and several other packages, such as **grImport** (Murrell, 2009) and **gridGraphviz** (Murrell, 2013), have also switched to using `makeContext()` and `makeContent()` methods, with further flow-on effects for **gridSVG**.

When to use `makeContent()` or `makeContext()`

The new functions in **grid** for R 3.0.0 are only *necessary* when it is not possible to determine either the drawing context or the drawing content at creation time.

These functions *may* be used in other situations. The main text box example used in this article does not strictly require using `makeContent()` because an `editDetails()` method could be used instead. The `editDetails()` hook is called whenever a grob is modified by `grid.edit()`. In the main example, the box around the text label could be recreated if the text label is modified. So a developer could elect to use `makeContent()` in some cases just because it may be easier than writing an `editDetails()` method.

Another alternative is to use an `edits` slot on a grob. This works a bit like a panel function for **lattice** plots. The idea is that changes to the children of a gTree can be specified as part of the description of the gTree and then only applied at drawing time, using `applyEdits()`, once the children of the gTree have been created. This approach has actually been implemented for **grid** axis grobs, but has not proven popular and has not been implemented anywhere else. One way to look at the changes to **grid** for R 3.0.0 is as a replacement for that `edits` slot approach; one which will hopefully prove more popular because it has wider benefits.

One reason why a grob may wish to delay construction of its content until drawing time is because the grob is expected to undergo dramatic changes before drawing occurs. In this case, it is inefficient to update the contents of the grob every time it is modified; it is better to wait until drawing time and perform the construction only once. The `"gtable"` class from the **gtable** package is an example of this sort of grob.

These are only examples of situations that might motivate the use of `makeContext()` and `makeContent()`. In some cases, the decision will be forced, but in other cases the choice may be deliberate, so there is no fixed rule for when we might need to use these functions.

It is also important to remember that simpler options may exist because **grid** already delays many calculations until drawing time via the use of `gp` and `vp` slots on grobs and the use of units for locations and dimensions. While it would be wrong to characterise these functions as a "last resort", developers of new grob classes should think at least twice before deciding that they are the best solution.

[2] https://github.com/pmur002/gtable

Other hook functions

In addition to the superceded `drawDetails()`, `preDrawDetails()`, and `postDrawDetails()` hook functions, and the `editDetails()` hook that was mentioned in the previous section, there are several other hook functions in **grid**.

The `validDetails()` function is called whenever a grob is created or edited. This can be used to check that the slots of a grob contain valid values. It is unaffected by the changes to **grid** described in this article.

The `widthDetails()` function is called when a `"grobwidth"` unit is evaluated, to determine the width of a grob. There is a similar function `heightDetails()`, plus `xDetails()` and `yDetails()` for determining locations on the boundary of a grob. Although these functions are not directly affected by the changes to **grid**, there are likely to be opportunities for code sharing between methods for these functions and `makeContent()` methods because grobs that have to calculate what to draw at drawing time are likely to have to also calculate what to draw in order to determine widths and heights or locations on a grob boundary.

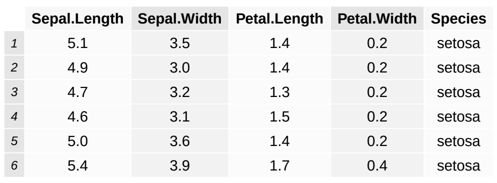
Who else could benefit?

A number of packages besides **ggplot2** and **gridSVG** build on top of **grid** and therefore could take advantage of the new changes to **grid**. One example is version 0.9.1 of the **gridExtra** package (Auguie, 2012).

```
> library(gridExtra)
```

The following code uses the `grid.table()` function from **gridExtra** to draw a tabular arrangement of values from a data frame.

```
> grid.newpage()
> grid.table(head(iris),
+            v.even.alpha=0.3, v.odd.alpha=1)
```



	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

The `grid.table()` function creates a `"table"` grob with a `drawDetails()` method that determines how to arrange the contents of the table only at drawing time. Because of this, none of the grobs that represent the actual table content are accessible. The call to `grid.ls()` below only reveals the overall `"table"` grob, but no other grobs, so there is no way to access or modify the contents of the table.

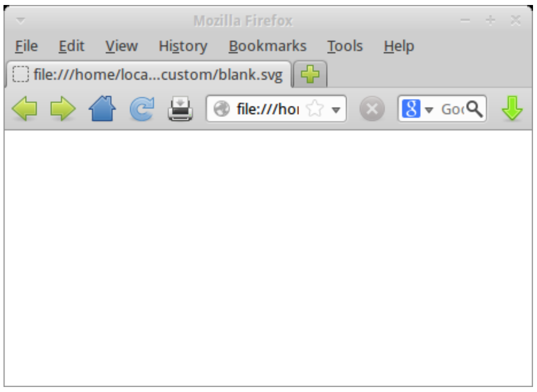
```
> grid.ls()
```

```
GRID.table.2
```

A further consequence is that the **gridSVG** package, which attempts to transform all grobs in the current scene to SVG, has nothing to transform; the code below produces a blank SVG image.

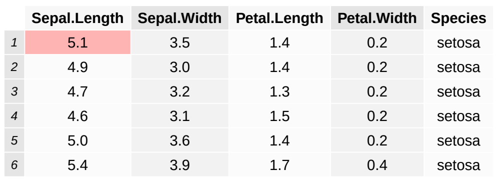
```
> library(gridSVG)
```

```
> gridToSVG("blank.svg")
```


Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///home/loca...custom/blank.svg
file:///hor
Goo

If there was a makeContent() method for "table" grobs, instead of a drawDetails() method, the function grid.table() would produce the same drawing, but it would become possible to grid.force() a "table" to make the grobs representing the content of the table accessible. The following code shows an example of how this could work, with grid.edit() used to modify the background colour for one of the cells in the table.

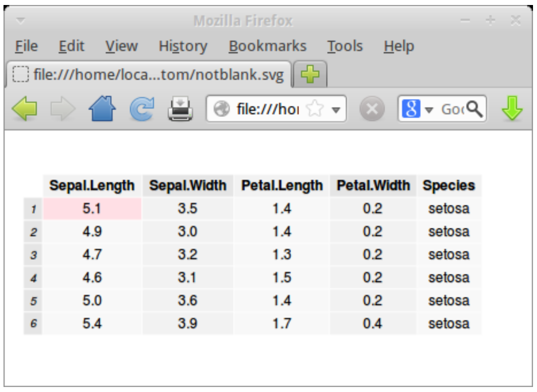
```
> grid.newpage()
> grid.table(head(iris),
+            v.even.alpha=0.3, v.odd.alpha=1)
> grid.force()
> grid.edit("core-fill-1", gp=gpar(fill="red"))
```



	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

Furthermore, export to SVG via the **gridSVG** package would now work, as shown below.

```
> gridToSVG("notblank.svg")
```


Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///home/loca...tom/notblank.svg
file:///hor
Goo

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

Summary

The functions makeContext() and makeContent() provide a new approach to developing a new **grid** grob class (replacing the old approach based on drawDetails() and preDrawDetails()).

The advantage of the new approach is that, for grobs that generate content at drawing time, it is possible to access and edit the low-level content that is generated at drawing time by calling the grid.force() function.

Together, these new features allow for greater flexibility in the development of new **grid** grob classes and greater powers to access and modify the low-level details of a **grid** scene.

Availability

The new **grid** functions `makeContext()`, `makeContent()`, `grid.force()`, and `grid.revert()` are only available from R version 3.0.0.

A number of higher-level graphics packages can potentially make use of these new facilities in **grid**, but it may take time for the respective package maintainers to make the necessary changes (if they elect to do so at all). For example, versions 0.9-0 of the **grImport** package and version 1.3-0 of the **gridSVG** package on R-Forge (Theußl and Zeileis, 2009) have incorporated changes. An example where future development may occur is `"gtable"` grobs from the **gtable** package, if changes from the fork on github are merged back into the original package.

Further reading

A more technical document describing the development and testing of these changes is available from the R developer web site (`http://www.stat.auckland.ac.nz/~paul/R/customGridRedesign.pdf`).

An earlier version of this article appeared as Technical Report 2012-9 on the Statistics Technical Blog of the Department of Statistics at the University of Auckland (Murrell, 2012).

Acknowledgements

Thanks to the reviewers of early versions of this article, who provided many useful suggestions that improved and expanded the article. Thanks also to the developers of **gtable** who provided the nudge to finally tackle the **grid** axis problem properly.

Bibliography

B. Auguie. *gridExtra: Functions in grid graphics*, 2012. URL `http://CRAN.R-project.org/package=gridExtra`. R package version 0.9.1. [p159]

P. Murrell. Importing vector graphics: The grImport package for R. *Journal of Statistical Software*, 30(4): 1–37, 2009. URL `http://www.jstatsoft.org/v30/i04/`. [p158]

P. Murrell. *R Graphics*. CRC Press, 2 edition, 6 2011. ISBN 9781439831762. [p149]

P. Murrell. Writing grid extensions. Technical Report 2012-9, Department of Statistics, The University of Auckland, http://stattech.wordpress.fos.auckland.ac.nz/2012-9-writing-grid-extensions/, 2012. [p161]

P. Murrell. *gridGraphviz: Drawing graphs with grid*, 2013. URL `http://CRAN.R-project.org/package=gridGraphviz`. R package version 0.2. [p158]

P. Murrell and S. Potter. *gridSVG: Export grid graphics as SVG*, 2013. R package version 1.3-0. [p158]

D. Sarkar. *Lattice: Multivariate Data Visualization with R*. Springer, New York, 2008. URL `http://lmdvr.r-forge.r-project.org`. ISBN 978-0-387-75968-5. [p149]

S. Theußl and A. Zeileis. Collaborative Software Development Using R-Forge. *The R Journal*, 1(1):9–14, May 2009. URL `http://journal.r-project.org/2009-1/RJournal_2009-1_Theussl+Zeileis.pdf`. [p161]

H. Wickham. *ggplot2: Elegant graphics for data analysis*. Springer New York, 2009. ISBN 978-0-387-98140-6. URL `http://had.co.nz/ggplot2/book`. [p149]

H. Wickham. *gtable: Arrange grobs in tables.*, 2012. URL `http://CRAN.R-project.org/package=gtable`. R package version 0.1.2. [p158]

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

R Foundation News

by Kurt Hornik

Donations and new members

Donations

Faculty of Psychology, University of Barcelona, Spain
Joanne M Potts, The Analytical Edge, Australia
Nassim Haddad, Belgium
Transmitting Science, Spain

New supporting institutions

Universität für Bodenkultur, Austria

New supporting members

Robin Crocket, UK
Tarundeep Dhot, Canada
Jan Marvin Garbuszus, Germany

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

News from the Bioconductor Project

Bioconductor Team
Program in Computational Biology
Fred Hutchinson Cancer Research Center

Bioconductor 2.13 was released on 15 October 2013. It is compatible with R 3.0.2, and consists of 749 software packages, 179 experiment data packages, and more than 690 up-to-date annotation packages. The release includes 84 new software packages, and enhancements to many others. Descriptions of new packages and updated NEWS files provided by current package maintainers are at http://bioconductor.org/news/bioc_2_13_release/.

Start using Bioconductor and R version 3.0.2 with

```
> source("http://bioconductor.org/biocLite.R")
> biocLite()
```

Install additional packages, e.g., **VariantTools**, with

```
> source("http://bioconductor.org/biocLite.R")
> biocLite("VariantTools")
```

Upgrade installed packages with

```
> source("http://bioconductor.org/biocLite.R")
> biocLite()
```

Explore available Bioconductor packages at http://bioconductor.org/packages/release/. All packages are grouped by 'BiocViews' to identify coherent groups of packages. Each package has an html page with the descriptions and links to vignettes, reference manuals, and use statistics.

A Bioconductor Amazon Machine Instance is available and updated; see http://bioconductor.org/help/bioconductor-cloud-ami.

Core Bioconductor packages

GenomicRanges and related packages continue to provide an extensive, mature and extensible framework for interacting with high throughput sequence data. Many contributed packages rely on this infrastructure for interoperable, re-usable analysis; Lawrence et al. (2013) provide an introduction.

Our large collection of microarray, transcriptome and organism-specific annotation packages have also been updated to include current information. Most of these packages now provide access to the 'select' interface (keys, columns, keytypes and select) which enable programmatic access to the databases they contain. This uniform interface simplifies the user experience; one illustration of this is in packages such as **Homo.sapiens**, which coordinate access to model organism gene, transcript, and pathway data bases. The **AnnotationHub**, now with more than 10,000 entries, complements our traditional offerings with diverse whole genome annotations from Ensembl, ENCODE, dbSNP, UCSC, and elsewhere.

This release includes the **PSICQUIC** package, an R interface to a powerful and standardized query language from the HUPO Proteomics Standard Initiative. More than two dozen well-known interaction databases are include, of which BioGrid, BIND, Reactome, STRING, IntAct, UniProt, DIP, and ChEMBL may be the best known. The query language uses a controlled vocabulary honed over several years by an active community, which returns documented and annotated interactions. One can now curate a detailed and up-to-date network illuminating functional relationships between genes and proteins of interest.

Other activities

In a change from the routine, our next Annual Meeting is in Boston, 30 July–1 August 2014, making it easier for east coast and European attendees. The 2013 Annual Meeting was in Seattle, 17–19 July, with our European developer community meeting in Cambridge, UK, 9–10 December. Our very active training and community events are advertised at `http://bioconductor.org/help/events/`. The Bioconductor mailing lists (`http://bioconductor.org/help/mailing-list/`) connect users with each other, to domain experts, and to maintainers eager to ensure that their packages satisfy the needs of leading edge approaches. Keep abreast of packages added to the 'devel' branch and other activities by following @Bioconductor on Twitter.

The past year marked our first participation in the Google Summer of Code project. We had many excellent applicants, and sponsored two projects to completion. GSoC participant Shawn Balcome, under Marc Carlson's mentoring, produced the **interactiveDisplay** package for Bioconductor / **shiny** integration. Michel Lang, mentored by Michael Lawrence, added very useful batch job and error recovery functionality to the **BiocParallel** package. This was a very successful experience!

Bibliography

M. Lawrence, W. Huber, H. Pagès, P. Aboyoun, M. Carlson, R. Gentleman, M. T. Morgan, and V. J. Carey. Software for computing and annotating genomic ranges. *PLoS Computational Biology*, 9(8):e1003118, 2013. doi: 10.1371/journal.pcbi.1003118. [p163]

Conference Report: Deuxièmes Rencontres R

by Aurélie Siberchicot and Stéphane Dray

Following the success of the first "Rencontres R" (Bordeaux, 2-3 July 2012, `http://r2012.bordeaux.inria.fr/`), the second meeting was held in Lyon on 27-28 June 2013. This French-speaking conference was a great success with 216 participants (110 were present at the first conference). The aim of the meeting was to provide a national forum for the exchange and sharing of ideas on the use of R in different disciplines. The number of participants and the list of sponsors (`http://r2013-lyon.sciencesconf.org/resource/sponsors`) demonstrate the increasing impact of R in both industry and academia in France. The program, detailed below, consisted of plenary sessions, oral presentations, lightning talks and poster sessions.

Invited speakers

Plenary sessions consisted of five talks (45 minutes) given by international experts:

- Hadley Wickham: *Visualising big data in R*
- Karim Chine: *R and the cloud*
- Jérôme Sueur: *R as a sound system*
- Yvonnick Noël: *A model comparison approach with* **R2STATS** *for teaching statistics in the social sciences*
- Gilbert Ritschard: **TraMineR***: a toolbox for exploring and rendering sequence data*

User-contributed sessions

Thirty-two users and developers presented original and attractive contributions covering nine fields:

- graphics & visualization
- high performance computing
- applied statistics
- data analysis & classification
- packages specific to a field of application
- teaching & pedagogy
- mixed models & longitudinal data
- dynamic documents & graphical interface
- genomic data analysis

Additionally, fourteen lightning talks (6 minutes with an automatic slide advancement) allowed participants to quickly present their work on R. Eight posters were presented during the poster session and the contribution of Ewen Gallic (University Rennes 1) entitled *Visualizing spatial processes using Ripley's correction* won the best poster prize (two R books).

Pre-conference tutorials

Two tutorial sessions were organized on Wednesday 26 June afternoon (around 30 participants per tutorial). The first session on *Parallel computing with R* was taught by Vincent Miele (University Lyon 1). During the second session, Anne-Béatrice Dufour (University Lyon 1) and Sylvain Mousset (University Lyon 1) showed *How to write a report with Sweave* and then Christophe Genolini (University Paris Ouest - Nanterre) introduced *How to create an R package*.

Conclusions

The organizing committee consisted of Julien Barnier (ENS Lyon), Stéphane Dray (Chairman, University Lyon 1), Kenneth Knoblauch (Inserm Lyon), Martyn Plummer (IARC) and Aurélie Siberchicot (University Lyon 1).

The program was designed by the scientific committee composed of Simon Barthelmé (University of Geneva), Marie-Laure Delignette-Muller (VetAgro Sup), Christophe Genolini, Robin Genuer (Chairman, University Bordeaux 2), Julie Josse (Agrocampus Rennes) and Jérôme Saracco (Bordeaux Institute of Technology).

The list of sponsors, participants, program, abstracts and slides are freely available on the conference web site at http://r2013-lyon.sciencesconf.org/. We would like to thank the members of the steering committee that initiated this conference: Marie Chavent (University Bordeaux 2), Robin Genuer, François Husson (Agrocampus Rennes), Julie Josse, Benoit Liquet (University Bordeaux 2) and Jérôme Saracco.

During the meeting, several discussions took place about the structuring of the French-speaking R-users community. After the conference, we launched an R-user group based at Lyon (see http://listes.univ-lyon1.fr/sympa/info/rlyon) and we will try to share some resources (common web site, etc.) with the existing semin-R group based at Paris (http://rug.mnhn.fr/semin-r). During the conference, it was also decided that the third edition of the "Rencontres R" will be organized in Montpellier in 2014.

Aurélie Siberchicot
Université de Lyon, F-69000, Lyon ; Université Lyon 1 ;
CNRS, UMR5558, Laboratoire de Biométrie et Biologie Evolutive,
F-69622, Villeurbanne, France.
aurelie.siberchicot@univ-lyon1.fr

Stéphane Dray
Université de Lyon, F-69000, Lyon ; Université Lyon 1 ;
CNRS, UMR5558, Laboratoire de Biométrie et Biologie Evolutive,
F-69622, Villeurbanne, France.
stephane.dray@univ-lyon1.fr

Changes in R

From 3.0.2 to 3.0.1

by the R Core Team

CHANGES IN R 3.0.2 patched

NEW FEATURES

- On Windows there is support for making '.texi' manuals using `texinfo` 5.0 or later: the setting is in file 'src/gnuwin32/MkRules.dist'.

  A packaging of the Perl script and modules for `texinfo` 5.2 has been made available at `http://www.stats.ox.ac.uk/pub/Rtools/`.

- `write.table()` now handles matrices of 2^{31} or more elements, for those with large amounts of patience and disc space.

- There is a new function, `La_version()`, to report the version of LAPACK in use.

- The HTML version of 'An Introduction to R' now has links to PNG versions of the figures.

- There is some support to produce manuals in ebook formats. (See 'doc/manual/Makefile'. Suggested by Mauro Cavalcanti.)

PACKAGE INSTALLATION

- The new field '`SysDataCompression`' in the 'DESCRIPTION' file allows user control over the compression used for 'sysdata.rda' objects in the lazy-load database.

C-LEVEL FACILITIES

- The long undocumented remapping of `rround()` to `Rf_fround()` in header 'Rmath.h' is now formally deprecated: use `fround()` directly.

- Remapping of `prec()` and `trunc()` in the 'Rmath.h' header has been disabled in C++ code (it has caused breakage with `libc++` headers).

BUG FIXES

- `getParseData()` truncated the imaginary part of complex number constants. (Reported by Yihui Xie.)

- `dbeta(x,a,b)` with `a` or `b` within a factor of 2 of the largest representable number could infinite-loop. (Reported by Ioannis Kosmidis.)

- `provideDimnames()` failed for arrays with a 0 dimension. (PR#15465)

- `rbind()` and `cbind()` did not handle list objects correctly. (PR#15468)

- `replayPlot()` now checks if it is replaying a plot from the same session.

- `rasterImage()` and `grid.raster()` now give error on an empty (zero-length) raster. (Reported by Ben North.)

- `plot.lm()` would sometimes scramble the labels in plot type 5. (PR#15458 and PR#14837)

- min() did not handle NA_character_ values properly. (Reported by Magnus Thor Torfason.)
- (Windows only.) readRegistry() would duplicate default values for keys. (PR#15455)
- str(...,strict.width = "cut") did not handle it properly when more than one line needed to be cut. (Reported by Gerrit Eichner.)
- Removing subclass back-references when S4 classes were removed or their namespace unloaded had several bugs (e.g., PR#15481).
- aggregate() could fail when there were too many levels present in the by argument. (PR#15004)
- namespaceImportFrom() needed to detect primitive functions when checking for duplicated imports (reported by Karl Forner).
- getGraphicsEvent() did not exit when a user closed the graphics window. (PR#15208)
- Errors in vignettes were not always captured and displayed properly. (PR#15495)
- contour() could fail when dealing with extremely small z values. (PR#15454)
- Several functions did not handle zero-length vectors properly, including browseEnv(), format(), gl(), relist() and summary.data.frame(). (E.g., PR#15499)
- Sweave() did not restore the R output to the console if it was interrupted by a user in the middle of evaluating a code chunk. (Reported by Michael Sumner.)
- Fake installs of packages with vignettes work again.
- Illegal characters in the input caused parse() (and thus source()) to segfault. (PR#15518)
- The nonsensical use of nmax = 1 in duplicated() or unique() is now silently ignored.
- qcauchy(p,*) is now fully accurate even when p is very close to 1. (PR#15521)
- The validmu() and valideta() functions in the standard glm() families now also report non-finite values, rather than failing.
- Saved vignette results (in a '.Rout.save' file) were not being compared to the new ones during R CMD check.
- Double-clicking outside of the list box (e.g. on the scrollbar) of a Tk listbox widget generated by tk_select.list() no longer causes the window to close. (PR#15407)
- Improved handling of edge cases in parallel::splitindices(). (PR#15552)
- HTML display of results from help.search() and ?? sometimes contained badly constructed links.
- c() and related functions such as unlist() converted raw vectors to invalid logical vectors. (PR#15535)
- (Windows only) When a call to system2() specified one of stdin, stdout or stderr to be a file, but the command was not found (e.g. it contained its arguments, or the program was not on the PATH), it left the file open and unusable until R terminated. (Reported by Mathew McLean.)
- The bmp() device was not recording res = NA correctly: it is now recorded as 72 ppi.
- Several potential problems with compiler-specific behaviour have been identified using the 'Undefined Behaviour Sanitizer' in conjunction with the clang compiler.

- `hcl()` now honours `NA` inputs (previously they were mapped to black).
- Some translations in base packages were being looked up in the main catalog rather than that for the package.
- As a result of the 3.0.2 change about 'the last second before the epoch', most conversions which should have given `NA` returned that time. (The platforms affected include Linux and OS X, but not Windows nor Solaris.)
- `rowsum` has more support for matrices and dataframes with 2^{31} or more elements. (PR#15587)
- `predict(<lm object>,interval = "confidence",scale = <something>)` now works. (PR#15564)
- The bug fix in 3.0.2 for PR#15411 was too aggressive, and sometimes removed spaces that should not have been removed. (PR#15583)
- Running R code in a **tcltk** callback failed to set the busy flag, which will be needed to tell OS X not to 'App Nap'.

CHANGES IN R 3.0.2

NEW FEATURES

- The 'NEWS' files have been re-organized.

  This file contains news for R >= 3.0.0: news for the 0.x.y, 1.x.y and 2.x.y releases is in files 'NEWS.0', 'NEWS.1' and 'NEWS.2'. The latter files are now installed when R is installed. An HTML version of news from 2.10.0 to 2.15.3 is available as 'doc/html/NEWS.2.html'.
- `sum()` for integer arguments now uses an integer accumulator of at least 64 bits and so will be more accurate in the very rare case that a cumulative sum exceeds 2^{53} (necessarily summing more than 4 million elements).
- The `example()` and `tools::Rd2ex()` functions now have parameters to allow them to ignore `\dontrun` markup in examples. (Suggested by Peter Solymos.)
- `str(x)` is considerably faster for very large lists, or factors with 100,000 levels, the latter as in PR#15337.
- `col2rgb()` now converts factors to character strings not integer codes (suggested by Bryan Hanson).
- `tail(warnings())` now works, via the new `‘[‘` method.
- There is now support for the LaTeX style file 'zi4.sty' which has in some distributions replaced 'inconsolata.sty'.
- `unlist(x)` now typically returns all non-list `x`s unchanged, not just the "vector" ones. Consequently, `format(lst)` now also works when the list `lst` has non-vector elements.
- The `tools::getVignetteInfo()` function has been added to give information about installed vignettes.
- New `assertCondition()`, etc. utilities in **tools**, useful for testing.
- Profiling now records non-inlined calls from byte-compiled code to `BUILTIN` functions.
- Various functions in **stats** and elsewhere that use non-standard evaluation are now more careful to follow the namespace scoping rules. E.g. `stats::lm()` can now find `stats::model.frame()` even if **stats** is not on the search path or if some package defines a function of that name.

- If an invalid/corrupt .Random.seed object is encountered in the workspace it is ignored with a warning rather than giving an error. (This allows R itself to rely on a working RNG, e.g. to choose a random port.)
- seq() and seq.int() give more explicit error messages if called with invalid (e.g. NaN) inputs.
- When parse() finds a syntax error, it now makes partial parse information available up to the location of the error. (Request of Reijo Sund.)
- Methods invoked by NextMethod() had a different dynamic parent to the generic. This was causing trouble where S3 methods invoked via lazy evaluation could lose track of their generic. (PR#15267)
- Code for the negative binomial distribution now treats the case size == 0 as a one-point distribution at zero.
- abbreviate() handles without warning non-ASCII input strings which require no abbreviation.
- read.dcf() no longer has a limit of 8191 bytes per line. (Wish of PR#15250.)
- formatC(x) no longer copies the class of x to the result, to avoid misuse creating invalid objects as in PR#15303. A warning is given if a class is discarded.
- Dataset npk has been copied from **MASS** to allow more tests to be run without recommended packages being installed.
- The initialization of the regression coefficients for non-degenerate differenced models in arima() has been changed and in some examples avoids a local maximum. (PR#15396)
- termplot() now has an argument transform.x to control the display of individual terms in the plot. (PR#15329)
- format() now supports digits = 0, to display nsmall decimal places.
- There is a new read-only par() parameter called "page", which returns a logical value indicating whether the next plot.new() call will start a new page.
- Processing Sweave and Rd documents to PDF now renders backticks and single quotes better in several instances, including in '\code' and '\samp' expressions.
- utils::modifyList() gets a new argument keep.null allowing NULL components in the replacement to be retained, instead of causing corresponding components to be deleted.
- tools::pkgVignettes() gains argument check; if set to TRUE, it will warn when it appears a vignette requests a non-existent vignette engine.

UTILITIES

- R CMD check --as-cran checks the line widths in usage and examples sections of the package Rd files.
- R CMD check --as-cran now implies '--timings'.
- R CMD check looks for command gfile if a suitable file is not found. (Although file is not from GNU, OpenCSW on Solaris installs it as gfile.)
- R CMD build (with the internal tar) checks the permissions of 'configure' and 'cleanup' files and adds execute permission to the recorded permissions for these files if needed, with a warning. This is useful on OSes and file systems which do not support execute permissions (notably, on Windows).

- `R CMD build` now weaves and tangles all vignettes, so suggested packages are not required during package installation if the source tarball was prepared with current `R CMD build`.
- `checkFF()` (used by `R CMD check`) does a better job of detecting calls from other packages, including not reporting those where a function has been copied from another namespace (e.g. as a default method). It now reports calls where `.NAME` is a symbol registered in another package.
- On Unix-alike systems, `R CMD INSTALL` now installs packages group writably whenever the library (`lib.loc`) is group writable. Hence, `update.packages()` works for other group members (suggested originally and from a patch by Dirk Eddelbuettel).
- `R CMD javareconf` now supports the use of symbolic links for `JAVA_HOME` on platforms which have `realpath`. So it is now possible to use

  ```
  R CMD javareconf JAVA_HOME=/usr/lib/jvm/java-1.7.0
  ```

  on a Linux system and record that value rather than the frequently-changing full path such as '/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.25.x86_64'.
- (Windows only.) `Rscript -e` requires a non-empty argument for consistency with Unix versions of R. (Also `Rterm -e` and `R -e`.)
- `R CMD check` does more thorough checking of declared packages and namespaces. It reports
  - packages declared in more than one of the '`Depends`', '`Imports`', '`Suggests`' and '`Enhances`' fields of the 'DESCRIPTION' file.
  - namespaces declared in '`Imports`' but not imported from, neither in the 'NAMESPACE' file nor using the `::` nor `:::` operators.
  - packages which are used in `library()` or `requires()` calls in the R code but were already put on the search path *via* '`Depends`'.
  - packages declared in '`Depends`' not imported *via* the 'NAMESPACE' file (except the standard packages). Objects used from '`Depends`' packages should be imported to avoid conflicts and to allow correct operation when the namespace is loaded but not attached.
  - objects imported *via* `:::` calls where `::` would do.
  - objects imported by `::` which are not exported.
  - objects imported by `:::` calls which do not exist.

  See 'Writing R Extensions' for good practice.
- `R CMD check` optionally checks for non-standard top-level files and directories (which are often mistakes): this is enabled for '`--as-cran`'.
- LaTeX style file `upquote.sty` is no longer included (the version was several years old): it is no longer used in R. A much later version is commonly included in LaTeX distributions but does not play well with the ae fonts which are the default for Sweave vignettes.
- `R CMD build` makes more use of the 'build' sub-directory of package sources, for example to record information about the vignettes.
- `R CMD check` analyses `:::` calls.

INSTALLATION and INCLUDED SOFTWARE

- The macros used for the texinfo manuals have been changed to work better with the incompatible changes made in `texinfo 5.x`.
- The minimum version for a system xz library is now 5.0.3 (was 4.999). This is in part to avoid 5.0.2, which can compress in ways other versions cannot decompress.
- The included version of PCRE has been updated to 8.33.
- The included version of `zlib` has been updated to 1.2.8, a bug-fix release.
- The included version of xz utils's `liblzma` has been updated to 5.0.5.
- Since `javareconf` (see above) is used when R is installed, a stable link for `JAVA_HOME` can be supplied then.
- Configuring with '`--disable-byte-compilation`' will override the 'DESCRIPTION' files of recommended packages, which typically require byte-compilation.
- More of the installation and checking process will work even when `TMPDIR` is set to a path containing spaces, but this is not recommended and external software (such as `texi2dvi`) may fail.

PACKAGE INSTALLATION

- Installation is aborted immediately if a `LinkingTo` package is not installed.
- `R CMD INSTALL` has a new option `--no-byte-compile` which will override a '`ByteCompile`' field in the package's 'DESCRIPTION' file.
- License '`BSD`' is deprecated: use '`BSD_3_clause`' or '`BSD_2_clause`' instead.

  License '`X11`' is deprecated: use '`MIT`' or '`BSD_2_clause`' instead.
- Version requirements for `LinkingTo` packages are now recognized: they are checked at installation. (Fields with version requirements were previously silently ignored.)
- The limit of 500 `S3method` entries in a `NAMESPACE` file has been removed.
- The default 'version' of Bioconductor for its packages has been changed to the upcoming '`2.13`', but this can be set by the environment variable `R_BIOC_VERSION`, e.g. in file 'Renviron.site'.

C-LEVEL FACILITIES

- 'Rdefines.h' has been tweaked so it can be included in C++ code after 'R_ext/Boolean.h' (which is included by 'R.h').

  Note that 'Rdefines.h' is not kept up-to-date, and 'Rinternals.h' is preferred for new code.
- `eval` and `applyClosure` are now protected against package code supplying an invalid `rho`.

DEPRECATED AND DEFUNCT

- The unused `namespace` argument to `package.skeleton()` is now formally deprecated and will be removed in R 3.1.0.
- `plclust()` is deprecated: use the `plot()` method for class `"hclust"` instead.
- Functions `readNEWS()` and `checkNEWS()` in package **tools** are deprecated (and they have not worked with current 'NEWS' files for a long time).

DOCUMENTATION

- 'An Introduction to R' has a new chapter on using R as a scripting language including interacting with the OS.

BUG FIXES

- `help.request()` could not determine the current version of R on CRAN. (PR#15241)
- On Windows, `file.info()` failed on root directories unless the path was terminated with an explicit `"."`. (PR#15302)
- The `regmatches<-()` replacement function mishandled results coming from `regexpr()`. (PR#15311)
- The help for `setClass()` and `representation()` still suggested the deprecated argument `representation=`. (PR#15312)
- `R CMD config` failed in an installed build of R 3.0.1 (only) when a sub-architecture was used. (Reported by Berwin Turlach.)
- On Windows, the installer modified the 'etc/Rconsole' and 'etc/Rprofile.site' files even when default options were chosen, so the MD5 sums did not refer to the installed versions. (Reported by Tal Galili.)
- `plot(hclust(),cex =)` respects `cex` again (and possibly others similarly). (Reported by Peter Langfelder.)
- If multiple packages were checked by `R CMD check`, and one was written for a different OS, it would set `--no-install` for all following packages as well as itself.
- `qr.coef()` and related functions did not properly coerce real vectors to complex when necessary. (PR#15332)
- `ftable(a)` now fixes up empty `dimnames` such that the result is printable.
- `package.skeleton()` was not starting its search for function objects in the correct place if `environment` was supplied. (Reported by Karl Forner.)
- Parsing code was changing the length field of vectors and confusing the memory manager. (PR#15345)
- The Fortran routine `ZHER2K` in the reference BLAS had a comment-out bug in two places. This caused trouble with `eigen()` for Hermitian matrices. (PR#15345 and report from Robin Hankin)
- `vignette()` and `browseVignettes()` did not display non-Sweave vignettes properly.
- Two warning/error messages have been corrected: the (optional) warning produced by a partial name match with a pairlist, the error message from a zero-length argument to the `:` operator. (Found by Radford Neal; PR#15358, PR#15356)
- `svd()` returned `NULL` rather than omitting components as documented. (Found by Radford Neal; PR#15360)
- `mclapply()` and `mcparallel()` with `silent = TRUE` could break a process that uses `stdout` output unguarded against broken pipes (e.g., `zip` will fail silently). To work around such issues, they now replace `stdout` with a descriptor pointed to '/dev/null' instead. For this purpose, internal `closeStdout` and `closeStderr` functions have gained the `to.null` flag.
- `log()`, `signif()` and `round()` now raise an error if a single named argument is not named `x`. (PR#15361)

- deparse() now deparses raw vectors in a form that is syntactically correct. (PR#15369)
- The jpeg driver in Sweave created a JPEG file, but gave it a '.png' extension. (PR#15370)
- Deparsing of infix operators with named arguments is improved. (PR#15350)
- mget(), seq.int() and numericDeriv() did not duplicate arguments properly. (PR#15352, PR#15353, PR#15354)
- kmeans(algorithm = "Hartigan-Wong") now always stops iterating in the QTran stage. (PR#15364).
- read.dcf() re-allocated incorrectly and so could segfault when called on a file with lines of more than 100 bytes.
- On systems where mktime() does not set errno, the last second before the epoch could not be converted from POSIXlt to POSIXct. (Reported by Bill Dunlap.)
- add1.glm() miscalculated F-statistics when df > 1. (Bill Dunlap, PR#15386).
- stem() now discards infinite inputs rather than hanging. (PR#15376)
- The parser now enforces C99 syntax for floating point hexadecimal constants (e.g. 0x1.1p0), rather than returning unintended values for malformed constants. (PR#15234)
- model.matrix() now works with very long LHS names (more than 500 bytes). (PR#15377)
- integrate() reverts to the pre-2.12.0 behaviour: from 2.12.0 to 3.0.1 it sometimes failed to achieve the requested tolerance and reported error estimates that were exceeded. (PR#15219)
- strptime() now handles '%W' fields with value 0. (PR#15915)
- R is now better protected against people trying to interact with the console in startup code. (PR#15325)
- Subsetting 1D arrays often lost dimnames (PR#15301).
- Unary + on a logical vector did not coerce to integer, although unary - did.
- na.omit() and na.exclude() added a row to a zero-row data frame. (PR#15399)
- All the (where necessary cut-down) vignettes are installed if R was configured with '--without-recommended-packages'.
- source() did not display filenames when reporting syntax errors.
- Syntax error reports misplaced the caret pointing out the bad token.
- (Windows only) Starting R with R (instead of Rterm or Rgui) would lose any zero-length strings from the command line arguments. (PR#15406)
- Errors in the encoding specified on the command line via --encoding=foo were not handled properly. (PR#15405)
- If x is a symbol, is.vector(x,"name") now returns TRUE, since "name" and "symbol" should be synonyms. (Reported by Hervé Pagès.)
- R CMD rtags works on platforms (such as OS X) with a XSI-conformant shell command echo. (PR#15231)
- is.unsorted(NA) returns false as documented (rather than NA).
- R CMD LINK did not know about sub-architectures.

- `system()` and `system2()` are better protected against users who misguidedly have spaces in the temporary directory path.
- `file.show()` and `edit()` are now more likely to work on file paths containing spaces. (Where external utilities are used, not the norm on Windows nor in `R.app` which should previously have worked.)
- Packages using the **methods** package are more likely to work when they import it but it is not attached. (Several parts of its C code were looking for its R functions on the search path rather than in its namespace.)
- `lgamma(-x)` is no longer `NaN` for very small `x`.
- (Windows) `system2()` now respects specifying `stdout` and `stderr` as files if called from `Rgui`. (PR#15393)
- Closing an `x11()` device whilst `locator()` or `identify()` is in progress no longer hangs R. (PR#15253)
- `list.dirs(full.names = FALSE)` was not implemented. (PR#15170)
- `format()` sometimes added unnecessary spaces. (PR#15411)
- `all.equal(check.names = FALSE)` would ignore the request to ignore the names and would check them as attributes.
- The symbol set by `tools::Rd2txt_options(itemBullet=)` was not respected in some locales. (PR#15435)
- `mcMap()` was not exported by package **parallel**. (PR#15439)
- `plot()` for `TukeyHSD` objects did not balance `dev.hold()` and `dev.flush()` calls on multi-page plots. (PR#15449)