

The Journal

Volume 6/1, June 2014

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

Contents


Editorial	3
---------------------	---

Contributed Research Articles

Taming PITCHf/x Data with XML2R and pitchRx	5
A Multiscale Test of Spatial Stationarity for Textured Images in R.	20
Stratified Weibull Regression Model for Interval-Censored Data	31
brainR : Interactive 3 and 4D Images of High Resolution Neuroimage Data	41
The RWiener Package: an R Package Providing Distribution Functions for the Wiener Diffusion Model	49
PivotalR : A Package for Machine Learning on Big Data	57
rotations : An R Package for $SO(3)$ Data.	68
ROSE : A Package for Binary Imbalanced Learning	79
investr : An R Package for Inverse Estimation	90
Rankcluster : An R Package for Clustering Multivariate Partial Rankings	101
The stringdist Package for Approximate String Matching	111
RStorm : Developing and Testing Streaming Algorithms in R.	123
The gridSVG Package	133
MRCV : A Package for Analyzing Categorical Variables with Multiple Response Op- tions	144
Archiving Reproducible Research with R and Dataverse.	151
oligoMask : A Framework for Assessing and Removing the Effect of Genetic Variants on Microarray Probes	159
sgr : A Package for Simulating Conditional Fake Ordinal Data	164

News and Notes

Web Technologies Task View.	178
Addendum to “Statistical Software from a Blind Person’s Perspective”	182
R Foundation News	183
News from the Bioconductor Project	184
Changes on CRAN	186
Changes in R	221

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

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

Editor-in-Chief:

Deepayan Sarkar

Editorial Board:

Hadley Wickham, Bettina Grün and Michael Lawrence

Editor Help Desk:

Uwe Ligges

Editor Book Reviews:

G. Jay Kerns

Department of Mathematics and Statistics

Youngstown State University

Youngstown, Ohio 44555-0002

USA

gkerns@ysu.edu

R Journal Homepage:

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

Email of editors and editorial board:

firstname.lastname@R-project.org

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

Editorial

by Deepayan Sarkar

On behalf of the editorial board, I am pleased to publish Volume 6, Issue 1 of the R Journal.

As usual, the bulk of the articles in this issue describe a variety of R packages, whose diversity reflects the ever increasing reach of R. Many of these packages make new data analysis methods available to the R community: **LS2Wstat** implements a test of spatial stationarity for textured images, **straweib** implements stratified Weibull regression models for interval-censored survival data, **rotations** provides various tools to work with rotation data, **ROSE** implements methods to deal with binary classification problems with high class imbalance, **investr** provides tools to solve inverse estimation problems for both linear and nonlinear regression models, **Rankcluster** enables model-based clustering of multivariate as well as partial rankings, **MRCV** allows analysis of data with multiple-response categorical variables, and **oligoMask** enables the removal of systematic effects of genetic variants when preprocessing microarray data. Several others fall into the category of more general purpose tools: **stringdist** implements various string distance functions and approximate string matching based on them, **RStorm** provides an environment to prototype and test streaming algorithms, **RWiener** implements distribution functions for the Wiener diffusion model that is useful for reaction time modeling, and **sgr** allows simulation of fake ordinal data to systematically study the effect of faked responses on inference. Two articles discuss packages that provide interfaces to other systems: **PivotalR** to various databases and the MADlib library for in-database machine learning, and **dvn** to The Dataverse Network to allow archival and sharing of reproducible research. Finally, we have three visualization packages that have quite different focus but share the common thread of interactive browser-based displays: **pitchRx** (along with **XML2R**) can be used to obtain and visualize basketball pitch data, **brainR** creates interactive displays of neuroimaging data, and **gridSVG** allows the exporting of grid graphics in the SVG format with several additional features.

The News and Notes section contains the usual updates on the R Foundation, the Bioconductor project, CRAN, and changes in R itself. In addition, we have a brief overview of the Web Technologies Task View available on CRAN. We also have a short addendum to an article published in the last issue of the R Journal, “Statistical Software from a Blind Person’s Perspective”, that provides a solution to a problem identified in the original article.

I hope you enjoy the issue.

Deepayan Sarkar

deepayan.sarkar@r-project.org

Taming PITCHf/x Data with XML2R and pitchRx

by Carson Sievert

Abstract XML2R is a framework that reduces the effort required to transform XML content into tables in a way that preserves parent to child relationships. **pitchRx** applies XML2R's grammar for XML manipulation to Major League Baseball Advanced Media (MLBAM)'s Gameday data. With **pitchRx**, one can easily obtain and store Gameday data in a remote database. The Gameday website hosts a wealth of XML data, but perhaps most interesting is PITCHf/x. Among other things, PITCHf/x data can be used to recreate a baseball's flight path from a pitcher's hand to home plate. With **pitchRx**, one can easily create animations and interactive 3D scatterplots of the baseball's flight path. PITCHf/x data is also commonly used to generate a static plot of baseball locations at the moment they cross home plate. These plots, sometimes called *strike-zone plots*, can also refer to a plot of event probabilities over the same region. **pitchRx** provides an easy and robust way to generate strike-zone plots using the **ggplot2** package.

Introduction

What is PITCHf/x?

PITCHf/x is a general term for a system that generates a series of 3D measurements of a baseball's path from a pitcher's hand to home plate (Alt and White, 2008).¹ In an attempt to estimate the location of the ball at any time point, a quadratic regression model with nine parameters (defined by the equations of motion for constant linear acceleration) is fit to each pitch. Studies with access to the actual measurements suggest that this model is quite reasonable – especially for non-knuckleball pitches (Nathan, 2008). That is, the fitted path often provides a reasonable estimate (within a couple of inches) of the actual locations. Unfortunately, only the parameter estimates are made available to the public. The website that provides these estimates is maintained by MLBAM and hosts a wealth of other baseball related data used to inform MLB's Gameday webcast service in near real time.

Why is PITCHf/x important?

On the business side of baseball, using statistical analysis to scout and evaluate players has become mainstream. When PITCHf/x was first introduced, (DiMeo, 2007) proclaimed it as,

“The new technology that will change statistical analysis [of baseball] forever.”

PITCHf/x has yet to fully deliver this claim, partially due to the difficulty in accessing and deriving insight from the large amount of complex information. By providing better tools to collect and visualize this data, **pitchRx** makes PITCHf/x analysis more accessible to the general public.

PITCHf/x applications

PITCHf/x data is and can be used for many different projects. It can also complement other baseball data sources, which poses an interesting database management problem. Statistical analysis of PITCHf/x data and baseball in general has become so popular that it has helped expose statistical ideas and practice to the general public. If you have witnessed television broadcasts of MLB games, you know one obvious application of PITCHf/x is locating pitches in the strike-zone as well as recreating flight trajectories, tracking pitch speed, etc. Some on-going statistical research related to PITCHf/x includes: classifying pitch types, predicting pitch sequences, and clustering pitchers with similar tendencies (Pane et al., 2013).

Contributions of pitchRx and XML2R

pitchRx has two main focuses (Sievert, 2014b). The first focus is to provide easy access to Gameday data. Not only is **pitchRx** helpful for collecting this data in bulk, but it has served as a helpful teaching and research aide (<http://baseballwithr.wordpress.com/> is one such example). Methods

¹A *pitcher* throws a ball to the opposing *batter*, who stands besides home plate and tries to hit the ball into the field of play.

for collecting Gameday data existed prior to **pitchRx**; however, these methods are not easily extensible and require juggling technologies that may not be familiar or accessible (Fast, 2007). Moreover, these working environments are less desirable than R for data analysis and visualization. Since **pitchRx** is built upon **XML2R**'s united framework, it can be easily modified and/or extended (Sievert, 2014a). For this same reason, **pitchRx** serves as a model for building customized XML data collection tools with **XML2R**.

The other main focus of **pitchRx** is to simplify the process creating popular PITCHf/x graphics. Strike-zone plots and animations made via **pitchRx** utilize the extensible **ggplot2** framework as well as various customized options (Wickham, 2009). **ggplot2** is a convenient framework for generating strike-zone plots primarily because of its facet schema which allows one to make visual comparisons across any combination of discrete variable(s). Interactive 3D scatterplots are based on the **rgl** package and useful for gaining a new perspective on flight trajectories (Adler et al.).

Getting familiar with Gameday

Gameday data is hosted and made available for free thanks to MLBAM via <http://gd2.mlb.com/components/game/mlb/>.² From this website, one can obtain many different types of data besides PITCHf/x. For example, one can obtain everything from structured media metadata to insider tweets. In fact, this website's purpose is to serve data to various <http://mlb.com> web pages and applications. As a result, some data is redundant and the format may not be optimal for statistical analysis. For these reasons, the scrape function is focused on retrieving data that is useful for PITCHf/x analysis and providing it in a convenient format for data analysis.

Navigating through the MLBAM website can be overwhelming, but it helps to recognize that a homepage exists for nearly every day and every game. For example, http://gd2.mlb.com/components/game/mlb/year_2011/month_02/day_26/ displays numerous hyperlinks to various files specific to February 26th, 2011. On this page is a hyperlink to [miniscoreboard.xml](#) which contains information on every game played on that date. This page also has numerous hyperlinks to game specific pages. For example, [gid_2011_02_26_phimlb_nyamlb_1/](#) points to the homepage for that day's game between the NY Yankees and Philadelphia Phillies. On this page is a hyperlink to the [players.xml](#) file which contains information about the players, umpires, and coaches (positions, names, batting average, etc.) coming into that game.

Starting from a particular game's homepage and clicking on the [inning/](#) directory, we *should* see another page with links to the [inning_all.xml](#) file and the [inning_hit.xml](#) file. If it is available, the [inning_all.xml](#) file contains the PITCHf/x data for that game. It's important to note that this file won't exist for some games, because some games are played in venues that do not have a working PITCHf/x system in place. This is especially true for preseason games and games played prior to the 2008 season when the PITCHf/x system became widely adopted.³ The [inning_hit.xml](#) files have manually recorded spatial coordinates of where a home run landed or where the baseball made initial contact with a defender after it was hit into play.

The relationship between these XML files and the tables returned by scrape is presented in Table 1. The pitch table is extracted from files whose name ends in [inning_all.xml](#). This is the only table returned by scrape that contains data on the pitch-by-pitch level. The atbat, runner, action and hip tables from this same file are commonly labeled somewhat ambiguously as play-by-play data. The player, coach, and umpire tables are extracted from [players.xml](#) and are classified as game-by-game since there is one record per person per game. Figure 1 shows how these tables can be connected to one another in a database setting. The direction of the arrows represent a one to possibly many relationship. For example, at least one pitch is thrown for each *at bat* (that is, each bout between pitcher and batter) and there are numerous at bats within each game.

In a rough sense, one can relate tables returned by scrape back to XML nodes in the XML files. For convenience, some information in certain XML nodes are combined into one table. For example, information gleaned from the 'top', 'bottom', and 'inning' XML nodes within [inning_all.xml](#) are included as [inning](#) and [inning_side](#) fields in the pitch, po, atbat, runner, and action tables. This helps reduce the burden of merging many tables together in order to have inning information on the play-by-play and/or pitch-by-pitch level. Other information is simply ignored simply because it is redundant. For example, the 'game' node within the [players.xml](#) file contains information that can be recovered from the game table extracted from the [miniscoreboard.xml](#) file. If the reader wants a more detailed explanation of fields in these tables, Marchi and Albert (2013) provide nice overview.

²Please be respectful of this service and store any information after you extract it instead of repeatedly querying the website. Before using any content from this website, please also read the [copyright](#).

³In this case, scrape will print "failed to load HTTP resource" in the R console (after the relevant file name) to indicate that no data was available.

Source file suffix	Information level	XML nodes	Tables returned by scrape
miniscoreboard.xml	game-by-game	games, game, game_media, media	game, media
players.xml	game-by-game	game, team, player, coach, umpire	player, coach, umpire
inning_all.xml	play-by-play, pitch-by-pitch	game, inning, bottom, top, atbat, po, pitch, runner, action	atbat, po, pitch, runner, action
inning_hit.xml	play-by-play	hitchart, hip	hip

Table 1: Structure of PITCHf/x and related Gameday data sources accessible to scrape

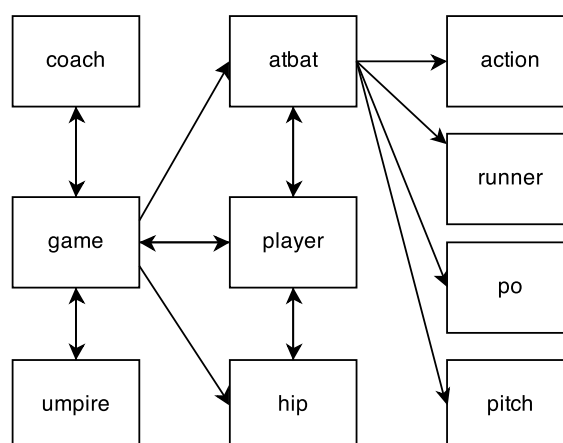


Figure 1: Table relations between Gameday data accessible via scrape. The direction of the arrows indicate a one to possibly many relationship.

Introducing XML2R

XML2R adds to the [CRAN Task View on Web Technologies and Services](#) by focusing on the transformation of XML content into a collection of tables. Compared to a lower-level API like the **XML** package, it can significantly reduce the amount of coding and cognitive effort required to perform such a task. In contrast to most higher-level APIs, it does not make assumptions about the XML structure or its source. Although **XML2R** works on any structure, performance and user experience are enhanced if the content has an inherent relational structure. **XML2R**'s novel approach to XML data collection breaks down the transformation process into a few simple steps and allows the user to decide how to apply those steps.

The next few sections demonstrate how **pitchRx** leverages **XML2R** in order to produce a collection of tables from `inning_all.xml` files. A similar approach is used by `pitchRx::scrape` to construct tables from the other Gameday files in Table 1. In fact, **XML2R** has also been used in the R package [bb scrapeR](#) which collects data from [nba.com](#) and [wnba.com](#).

Constructing file names

Sometimes the most frustrating part of obtaining data in bulk off of the web is finding the proper collection of URLs or file names of interest. Since files on the Gameday website are fairly well organized, the `makeUrls` function can be used to construct urls that point to every game's homepage within a window of dates.

```

urls <- makeUrls(start = "2011-06-01", end = "2011-06-01")
sub("http://gd2.mlb.com/components/game/mlb/", "", head(urls))

#> [1] "year_2011/month_06/day_01/gid_2011_06_01_anamlb_kcamlb_1"
#> [2] "year_2011/month_06/day_01/gid_2011_06_01_bamlb_seamlb_1"

```

```
#> [3] "year_2011/month_06/day_01/gid_2011_06_01_chamlb_bosmlb_1"
#> [4] "year_2011/month_06/day_01/gid_2011_06_01_clemlb_tormlb_1"
#> [5] "year_2011/month_06/day_01/gid_2011_06_01_colmlb_lanmlb_1"
#> [6] "year_2011/month_06/day_01/gid_2011_06_01_flomlb_arimlb_1"
```

Extracting observations

Once we have a collection of XML files, the next step is to parse the content into a list of *observations*. An observation is technically defined as a matrix with one row and some number of columns. The columns are defined by XML attributes and the XML value (if any) for a particular XML lineage. The name of each observation tracks the XML hierarchy so observations can be grouped together in a sensible fashion at a later point.

```
library(XML2R)
files <- paste0(urls, "/inning/inning_all.xml")
obs <- XML2Obs(files, url.map = TRUE, quiet = TRUE)
table(names(obs))

#>
#>
#> game game//inning
#> 2 18
#> game//inning//bottom//action game//inning//bottom//atbat
#> 13 69
#> game//inning//bottom//atbat//pitch game//inning//bottom//atbat//po
#> 247 4
#> game//inning//bottom//atbat//runner game//inning//top//action
#> 50 20
#> game//inning//top//atbat game//inning//top//atbat//pitch
#> 79 278
#> game//inning//top//atbat//po game//inning//top//atbat//runner
#> 17 89
#> url_map
#> 1
```

This output tells us that 247 pitches were thrown in the bottom inning and 278 were thrown in the top inning on June 1st, 2011. Also, there are 12 different levels of observations. The list element named `url_map` is not considered an observation and was included since `url.map = TRUE`. This helps avoid repeating long file names in the `url_key` column which tracks the mapping between observations and file names.

```
obs[1]

#> $`game//inning//top//atbat//pitch`
#> des id type tfs tfs_zulu x y
#> [1,] "Called Strike" "3" "S" "191018" "2011-06-01T23:10:18Z" "109.87" "145.06"
#> sv_id start_speed end_speed sz_top sz_bot pfx_x pfx_z px
#> [1,] "110601_191020" "87.9" "82.1" "3.6" "1.65" "-6.7" "4.36" "-0.213"
#> pz x0 y0 z0 vx0 vy0 vz0 ax
#> [1,] "2.611" "-1.612" "50.0" "5.633" "5.808" "-128.728" "-2.903" "-11.406"
#> ay az break_y break_angle break_length pitch_type
#> [1,] "22.954" "-24.681" "23.9" "22.5" "6.5" "SI"
#> type_confidence zone nasty spin_dir spin_rate cc mt url_key
#> [1,] ".798" "5" "39" "236.697" "1538.041" "" "" "url1"
```

Renaming observations

Before grouping observations into a collection tables based on their names, one may want to `re_name` observations. Observations with names `'game//inning//bottom//atbat'` and `'game//inning//top//atbat'` should be included in the same table since they share XML attributes (in other words, the observations share variables).

```
tmp <- re_name(obs, equiv = c("game//inning//top//atbat",
                              "game//inning//bottom//atbat"), diff.name = "inning_side")
```


By passing these names to the `equiv` argument, `re_name` determines the difference in the naming scheme and suppresses that difference. In other words, observation names that match the `equiv` argument will be renamed to 'game//inning//atbat'. The information removed from the name is not lost; however, as a new column is appended to the end of each relevant observation. For example, notice how the `inning_side` column contains the part of the name we just removed:

```
tmp[grep("game//inning//atbat", names(tmp))][1:2]

#> $`game//inning//atbat`
#>   num b   s   o   start_tfs start_tfs_zulu   batter   stand b_height
#> [1,] "1" "0" "1" "0" "190935" "2011-06-01T23:09:35Z" "430001" "R"   "5-10"
#>   pitcher p_throws
#> [1,] "502190" "R"
#>   des                                     event       score
#> [1,] "Rickie Weeks homers (10) on a fly ball to left field. " "Home Run" "T"
#>   home_team_runs away_team_runs url_key inning_side
#> [1,] "0"           "1"           "url1" "top"
#>
#> $`game//inning//atbat`
#>   num b   s   o   start_tfs start_tfs_zulu   batter   stand b_height
#> [1,] "2" "0" "0" "0" "191105" "2011-06-01T23:11:05Z" "460579" "L"   "6-0"
#>   pitcher p_throws
#> [1,] "502190" "R"
#>   des
#> [1,] "Nyjer Morgan triples (3) on a line drive to right fielder Jay Bruce. "
#>   event url_key inning_side
#> [1,] "Triple" "url1" "top"
```

For similar reasons, other observation name pairs are renamed in a similar fashion.

```
tmp <- re_name(tmp, equiv = c("game//inning//top//atbat//runner",
                             "game//inning//bottom//atbat//runner"), diff.name = "inning_side")
tmp <- re_name(tmp, equiv = c("game//inning//top//action",
                             "game//inning//bottom//action"), diff.name = "inning_side")
tmp <- re_name(tmp, equiv = c("game//inning//top//atbat//po",
                             "game//inning//bottom//atbat//po"), diff.name = "inning_side")
obs2 <- re_name(tmp, equiv = c("game//inning//top//atbat//pitch",
                              "game//inning//bottom//atbat//pitch"), diff.name = "inning_side")
table(names(obs2))

#>
#>               game               game//inning
#>               2                 18
#> game//inning//action game//inning//atbat
#>               33                148
#> game//inning//atbat//pitch game//inning//atbat//po
#>               525                 21
#> game//inning//atbat//runner url_map
#>               139                 1
```

Linking observations

After all that renaming, we now have 7 different levels of observations. Let's examine the first three observations on the `game//inning` level:

```
obs2[grep("^game//inning$", names(obs2))][1:3]

#> $`game//inning`
#>   num away_team home_team next url_key
#> [1,] "1" "mil"   "cin"    "Y"   "url1"
#>
#> $`game//inning`
#>   num away_team home_team next url_key
#> [1,] "2" "mil"   "cin"    "Y"   "url1"
#>
#> $`game//inning`
```

```
#>      num away_team home_team next url_key
#> [1,] "3" "mil"      "cin"      "Y"   "url1"
```

Before grouping observations into tables, it is usually important preserve the parent-to-child relationships in the XML lineage. For example, one may want to map a particular pitch back to the inning in which it was thrown. Using the `add_key` function, the relevant value of `num` for `game//inning` observations can be recycled to its XML descendants.

```
obskey <- add_key(obs2, parent = "game//inning", recycle = "num", key.name = "inning")
```

```
#> A key for the following children will be generated for the game//inning node:
#> game//inning//atbat//pitch
#> game//inning//atbat//runner
#> game//inning//atbat
#> game//inning//atbat//po
#> game//inning//action
```

As it turns out, the `away_team` and `home_team` columns are redundant as this information is embedded in the `url` column. Thus, there is only one other informative attribute on this level which is `next`. By recycling this value among its descendants, we remove any need to retain a `game//inning` table.

```
obskey <- add_key(obskey, parent = "game//inning", recycle = "next")
```

```
#> A key for the following children will be generated for the game//inning node:
#> game//inning//atbat//pitch
#> game//inning//atbat//runner
#> game//inning//atbat
#> game//inning//atbat//po
#> game//inning//action
```

It is also imperative that we can link a pitch, runner, or po back to a particular atbat. This can be done as follows:

```
obskey <- add_key(obskey, parent = "game//inning//atbat", recycle = "num")
```

```
#> A key for the following children will be generated for the game//inning//atbat node:
#> game//inning//atbat//pitch
#> game//inning//atbat//runner
#> game//inning//atbat//po
```

Collapsing observations

Finally, we are in a position to pool together observations that have a common name. The `collapse_obs` function achieves this by row binding observations with the same name together and returning a list of matrices. Note that `collapse_obs` does not require that observations from the same level to have the same set of variables in order to be bound into a common table. In the case where variables are missing, NAs will be inserted as values.

```
tables <- collapse_obs(obskey)
#As mentioned before, we do not need the 'inning' table
tables <- tables[!grepl("^game//inning$", names(tables))]
table.names <- c("game", "action", "atbat", "pitch", "po", "runner")
tables <- setNames(tables, table.names)
head(tables[["runner"]])

#>      id      start end  event      score rbi earned url_key inning_side
#> [1,] "430001" ""    ""    "Home Run" "T"    "T" "T"    "url1" "top"
#> [2,] "460579" ""    "3B"  "Triple"  NA     NA NA     "url1" "top"
#> [3,] "460579" "3B"  ""    "Groundout" "T"    "T" "T"    "url1" "top"
#> [4,] "425902" ""    "1B"  "Single"  NA     NA NA     "url1" "top"
#> [5,] "425902" "1B"  ""    "Pop Out"  NA     NA NA     "url1" "top"
#> [6,] "458015" ""    "1B"  "Single"  NA     NA NA     "url1" "bottom"
#>      inning next num
#> [1,] "1"      "Y"  "1"
#> [2,] "1"      "Y"  "2"
#> [3,] "1"      "Y"  "3"
#> [4,] "1"      "Y"  "4"
#> [5,] "1"      "Y"  "6"
#> [6,] "1"      "Y"  "9"
```

Collecting Gameday data with pitchRx

The main scraping function in **pitchRx**, `scrape`, can be used to easily obtain data from the files listed in Table 1. In fact, any combination of these files can be queried using the `suffix` argument. In the example below, the `start` and `end` arguments are also used so that all available file types for June 1st, 2011 are queried.

```
library(pitchRx)
files <- c("inning/inning_all.xml", "inning/inning_hit.xml",
          "miniscoreboard.xml", "players.xml")
dat <- scrape(start = "2011-06-01", end = "2011-06-01", suffix = files)
```

The `game.ids` option can be used instead of `start` and `end` to obtain an equivalent `dat` object. This option can be useful if the user wants to query specific games rather than all games played over a particular time span. When using this `game.ids` option, the built-in `gids` object, is quite convenient.

```
data(gids, package = "pitchRx")
gids11 <- gids[grepl("2011_06_01", gids)]
head(gids11)

#> [1] "gid_2011_06_01_anamlb_kcamlb_1" "gid_2011_06_01_balmlb_seamlb_1"
#> [3] "gid_2011_06_01_chamlb_bosmlb_1" "gid_2011_06_01_clemlb_tormlb_1"
#> [5] "gid_2011_06_01_colmlb_lanmlb_1" "gid_2011_06_01_flomlb_arimlb_1"

dat <- scrape(game.ids = gids11, suffix = files)
```

The object `dat` is a list of data frames containing all data available for June 1st, 2011 using `scrape`. The list names match the table names provided in Table 1. For example, `dat$atbat` is data frame with every at bat on June 1st, 2011 and `dat$pitch` has information related to the outcome of each pitch (including PITCHf/x parameters). The object `size` of `dat` is nearly 300MB. Multiplying this number by 100 days exceeds the memory of most machines. Thus, if a large amount of data is required, the user should exploit the R database interface ([R Special Interest Group on Databases, 2013](#)).

Storing and querying Gameday data

Since PITCHf/x data can easily exhaust memory, one should consider establishing a database instance before using `scrape`. By passing a database connection to the `connect` argument, `scrape` will try to create (and/or append to existing) tables using that connection. If the connection fails for some reason, tables will be written as csv files in the current working directory. The benefits of using the `connect` argument includes improved memory management which can greatly reduce run time. `connect` will support a MySQL connection, but creating a SQLite database is quite easy with **dplyr** ([Wickham and Francois, 2014](#)).

```
library(dplyr)
db <- src_sqlite("GamedayDB.sqlite3", create = TRUE)
# Collect and store all PITCHf/x data from 2008 to now
scrape(start = "2008-01-01", end = Sys.Date(),
       suffix = "inning/inning_all.xml", connect = db$con)
```

In the later sections, animations of four-seam and cut fastballs thrown by Mariano Rivera and Phil Hughes during the 2011 season are created. In order to obtain the data for those animations, one could query `db` which now has PITCHf/x data from 2008 to date. This query requires criteria on: the `pitcher_name` field (in the `atbat` table), the `pitch_type` field (in the `pitch` table), and the `date` field (in both tables). To reduce the time required to search those records, one should create an index on each of these three fields.

```
library(DBI)
dbSendQuery(db$con, "CREATE INDEX pitcher_index ON atbat(pitcher_name)")
dbSendQuery(db$con, "CREATE INDEX type_index ON pitch(pitch_type)")
dbSendQuery(db$con, "CREATE INDEX date_atbat ON atbat(date)")
```

As a part of our query, we'll have to join the `atbat` table together with the `pitch` table. For this task, the `gameday_link` and `num` fields are helpful since together they provide a way to match pitches with at bats. For this reason, a multi-column index on the `gameday_link` and `num` fields will further reduce run time of the query.

```
dbSendQuery(db$con, 'CREATE INDEX pitch_join ON pitch(gameday_link, num)')
dbSendQuery(db$con, 'CREATE INDEX atbat_join ON atbat(gameday_link, num)')
```

Although the query itself could be expressed entirely in SQL, **dplyr**'s grammar for data manipulation (which is database agnostic) can help to simplify the task. In this case, `at.bat` is a tabular *representation* of the remote `atbat` table restricted to cases where Rivera or Hughes was the pitcher. That is, `at.bat` does not contain the actual data, but it does contain the information necessary to retrieve it from the database.

```
at.bat <- tbl(db, "atbat") %>%
  filter(pitcher_name %in% c("Mariano Rivera", "Phil Hughes"))
```

Similarly, `fbs` is a tabular representation of the pitch table restricted to four-seam (FF) and cut fastballs (FC).

```
fbs <- tbl(db, "pitch") %>%
  filter(pitch_type %in% c("FF", "FC"))
```

An `inner_join` of these two filtered tables returns a tabular representation of all four-seam and cut fastballs thrown by Rivera and Hughes. Before `collect` actually performs the database query and brings the relevant data into the R session, another restriction is added so that only pitches from 2011 are included.

```
pitches <- inner_join(fbs, at.bat) %>%
  filter(date >= "2011_01_01" & date <= "2012_01_01") %>%
  collect()
```

Visualizing PITCHf/x

Strike-zone plots and umpire bias

Amongst the most common PITCHf/x graphics are strike-zone plots. Such a plot has two axes and the coordinates represent the location of baseballs as they cross home plate. The term strike-zone plot can refer to either *density* or *probabilistic* plots. Density plots are useful for exploring what *actually* occurred, but probabilistic plots can help address much more interesting questions using statistical inference. Although probabilistic plots can be used to visually track any event probability across the strike-zone, their most popular use is for addressing umpire bias in a strike versus ball decision [Green and Daniels \(2014\)](#). The probabilistic plots section demonstrates how **pitchRx** simplifies the process behind creating such plots via a case study on the impact of home field advantage on umpire decisions.

In the world of sports, it is a common belief that umpires (or referees) have a tendency to favor the home team. PITCHf/x provides a unique opportunity to add to this discussion by modeling the probability of a called strike at home games versus away games. Specifically, conditioned upon the umpire making a decision at a specific location in the strike-zone, if the probability that a home pitcher receives a called strike is higher than the probability that an away pitcher receives a called strike, then there is evidence to support umpire bias towards a home pitcher.

There are many different possible outcomes of each pitch, but we can condition on the umpire making a decision by limiting to the following two cases. A *called strike* is an outcome of a pitch where the batter does not swing and the umpire declares the pitch a strike (which is a favorable outcome for the pitcher). A *ball* is another outcome where the batter does not swing and the umpire declares the pitch a ball (which is a favorable outcome for the batter). All decisions made between 2008 and 2013 can be obtained from `db` with the following query using **dplyr**.

```
# First, add an index on the pitch description to speed up run-time
dbSendQuery(db$con, "CREATE INDEX des_index ON pitch(des)")

pitch <- tbl(db, "pitch") %>%
  filter(des %in% c("Called Strike", "Ball")) %>%
  # Keep pitch location, descriptions
  select(px, pz, des, gameday_link, num) %>%
  # 0-1 indicator of strike/ball
  mutate(strike = as.numeric(des == "Called Strike"))

atbat <- tbl(db, "atbat") %>%
  # Select variables to be used later as covariates in probabilistic models
```

```

select(b_height, p_throws, stand, inning_side, date, gameday_link, num)

decisions <- inner_join(pitch, atbat) %>%
  filter(date <= "2014_01_01") %>%
  collect()

```

Density plots

The decisions data frame contains data on over 2.5 million pitches thrown from 2008 to 2013. About a third of them are called strikes and two-thirds balls. Figure 2 shows the density of all called strikes. Clearly, most called strikes occur on the outer region of the strike-zone. Many factors could contribute to this phenomenon; which we will not investigate here.

```

# strikeFX uses the stand variable to calculate strike-zones
# Here is a slick way to create better facet titles without changing data values
relabel <- function(variable, value) {
  value <- sub("^R$", "Right-Handed Batter", value)
  sub("^L$", "Left-Handed Batter", value)
}
strikes <- subset(decisions, strike == 1)
strikeFX(strikes, geom = "raster", layer = facet_grid(. ~ stand, labeller = relabel))

```

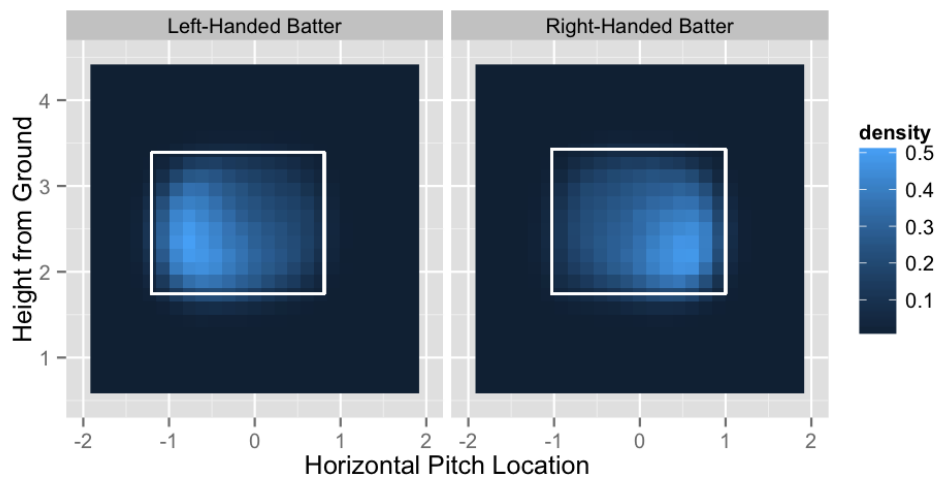


Figure 2: Density of called strikes for right-handed batters and left-handed batters (from 2008 to 2013).

Figure 2 shows one static rectangle (or strike-zone) per plot automatically generated by strikeFX. The definition of the strike-zone is notoriously ambiguous. As a result, the boundaries of the strike-zone may be noticeably different in some situations. However, we can achieve a fairly accurate representation of strike-zones using a rectangle defined by batters' average height and stance (Fast, 2011). As Figure 4 reinforces, batter stance makes an important difference since the strike-zone seems to be horizontally shifted away from the batter. The batter's height is also important since the strike-zone is classically defined as approximately between the batter's knees and armpits.

Figure 2 has is one strike-zone per plot since the layer option contains a **ggplot2** argument that facets according to batter stance. Facet layers are a powerful tool for analyzing PITCHf/x data because they help produce quick and insightful comparisons. In addition to using the layer option, one can add layers to a graphic returned by strikeFX using **ggplot2** arithmetic. It is also worth pointing out that Figure 2 could have been created without introducing the strikes data frame by using the density1 and density2 options.

```

strikeFX(decisions, geom = "raster", density1 = list(des = "Called Strike"),
  density2 = list(des = "Called Strike")) + facet_grid(. ~ stand, labeller = relabel)

```

In general, when density1 and density2 are identical, the result is equivalent to subsetting the data frame appropriately beforehand. More importantly, by specifying *different* values for density1 and density2, differenced densities are easily generated. In this case, a grid of density estimates for

density2 are subtracted from the corresponding grid of density estimates for density1. Note that the default NULL value for either density option infers that the entire data set defines the relevant density. Thus, if density2 was NULL (when density1 = list(des = 'Called Strike')), we would obtain the density of called strikes minus the density of *both* called strikes and balls. In Figure 3, we define density1 as called strikes and define density2 as balls. As expected, we see positive density values (in blue) inside the strike-zone and negative density values (in red) outside of the strike-zone.

```
strikeFX(decisions, geom = "raster", density1 = list(des = "Called Strike"),
  density2 = list(des = "Ball"), layer = facet_grid(. ~ stand, labeller = relabel))
```

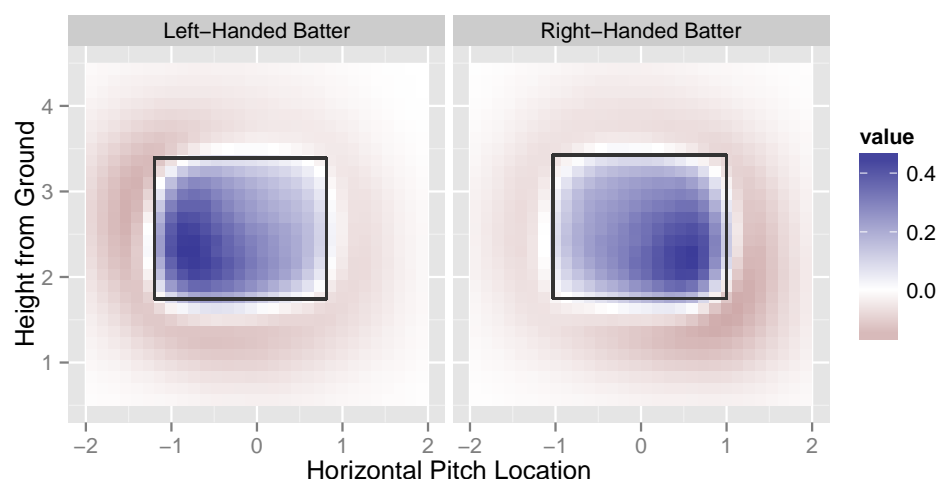


Figure 3: Density of called strikes minus density of balls for both right-handed batters and left-handed batters (from 2008 to 2013). The blue region indicates a higher frequency of called strikes and the red region indicates a higher frequency of balls.

These density plots are helpful for visualizing the observed frequency of events; however, they are not very useful for addressing our umpire bias hypothesis. Instead of looking simply at the *density*, we want to model the *probability* of a strike called at each coordinate given the umpire has to make a decision.

Probabilistic plots

There are many approaches to probabilistic modeling over a two dimensional spatial region. Since our response is often categorical, generalized additive models (GAMs) is a popular and desirable approach to modeling events over the strike-zone (Mills, 2010). There are numerous R package implementations of GAMs, but the bam function from the *mgcv* package has several desirable properties (Wood, 2006). Most importantly, the smoothing parameter can be estimated using several different methods. In order to have a reasonable estimate of the smooth 2D surface, GAMs require fairly large amount of observations. As a result, run time can be an issue – especially when modeling 2.5 million observations! Thankfully, the bam function has a cluster argument which allows one to distribute computations across multiple cores using the built in *parallel* package.

```
library(parallel)
cl <- makeCluster(detectedCores() - 1)
library(mgcv)
m <- bam(strike ~ interaction(stand, p_throws, inning_side) +
  s(px, pz, by = interaction(stand, p_throws, inning_side)),
  data = decisions, family = binomial(link = 'logit'), cluster = cl)
```

This formula models the probability of a strike as a function of the baseball's spatial location, the batter's stance, the pitcher's throwing arm, and the side of the inning. Since home pitchers always pitch during the top of the inning, inning_side also serves as an indication of whether a pitch is thrown by a home pitcher. In this case, the interaction function creates a factor with eight different levels since every input factor has two levels. Consequently, there are 8 different levels of smooth surfaces over the spatial region defined by px and pz.

The fitted model `m` contains a lot of information which `strikeFX` uses in conjunction with any `ggplot2` facet commands to infer which and how surfaces should be plotted. In particular, the `var.summary` is used to identify model covariates, as well their default conditioning values. In our case, the majority of decisions are from right-handed pitchers and the top of the inning. Thus, the default conditioning values are "top" for `inning_side` and "R" for `p_throws`. If different conditioning values are desired, `var.summary` can be modified accordingly. To demonstrate, Figure 4 shows 2 of the 8 possible surfaces that correspond to a right-handed *away* pitcher.

```
away <- list(inning_side = factor("bottom", levels = c("top", "bottom")))
m$var.summary <- modifyList(m$var.summary, away)
strikeFX(decisions, model = m, layer = facet_grid(. ~ stand, labeller = relabel))
```

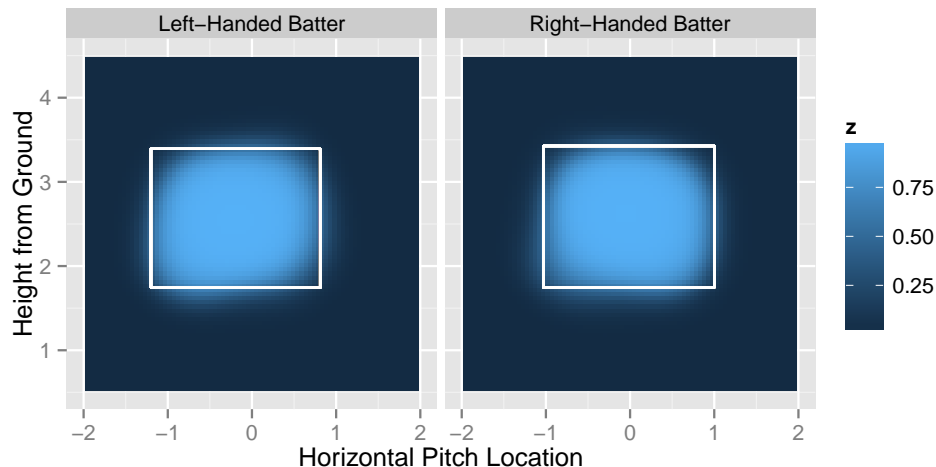


Figure 4: Probability that a right-handed away pitcher receives a called strike (provided the umpire has to make a decision). Plots are faceted by the handedness of the batter.

Using the same intuition exploited earlier to obtain differenced density plots, we can easily obtain differenced probability plots. To obtain Figure 5, we simply add `p_throws` as another facet variable and `inning_side` as a differencing variable. In this case, conditioning values do not matter since every one of the 8 surfaces are required in order to produce Figure 5.

```
# Function to create better labels for both stand and p_throws
relabel2 <- function(variable, value) {
  if (variable %in% "stand")
    return(sub("^L$", "Left-Handed Batter",
              sub("^R$", "Right-Handed Batter", value)))
  if (variable %in% "p_throws")
    return(sub("^L$", "Left-Handed Pitcher",
              sub("^R$", "Right-Handed Pitcher", value)))
}
strikeFX(decisions, model = m, layer = facet_grid(p_throws ~ stand, labeller = relabel2),
  density1 = list(inning_side = "top"), density2 = list(inning_side = "bottom"))
```

The four different plots in Figure 5 represent the four different combination of values among `p_throws` and `stand`. In general, provided that a pitcher throws to a batter in the blue region, the pitch is more likely to be called a strike if the pitcher is on their home turf. Interestingly, there is a well-defined blue elliptical band around the boundaries of the typical strike-zone. Thus, home pitchers are more likely to receive a favorable call – especially when the classification of the pitch is in question. In some areas, the home pitcher has up to a 6 percent higher probability of receiving a called strike than an away pitcher. The subtle differences in spatial patterns across the different values of `p_throws` and `stand` are interesting as well. For instance, pitching at home has a large positive impact for a left-handed pitcher throwing in the lower inside portion of the strike-zone to a right-handed batter, but the impact seems negligible in the mirror opposite case. Differenced probabilistic densities are clearly an interesting visual tool for analyzing PITCHfx data. With `strikeFX`, one can quickly and easily make all sorts of visual comparisons for various situations. In fact, one can explore and compare the probabilistic structure of any well-defined event over a strike-zone region (for example, the probability a batter reaches base) using a similar approach.

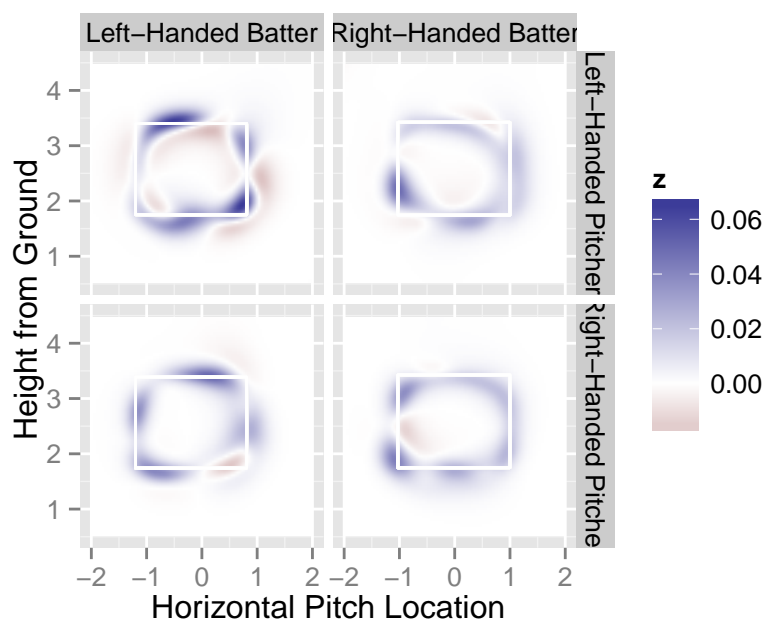


Figure 5: Difference between home and away pitchers in the probability of a strike (provided the umpire has to make a decision). The blue regions indicate a higher probability of a strike for home pitchers and red regions indicate a higher probability of a strike for away pitchers. Plots are faceted by the handedness of both the pitcher and the batter.

2D animation

animateFX provides convenient and flexible functionality for animating the trajectory of any desired set of pitches. For demonstration purposes, this section animates every four-seam and cut fastball thrown by Mariano Rivera and Phil Hughes during the 2011 season. These pitches provide a good example of how facets play an important role in extracting new insights. Similar methods can be used to analyze any MLB player (or combination of players) in greater detail.

animateFX tracks three dimensional pitch locations over a sequence of two dimensional plots. The animation takes on the viewpoint of the umpire; that is, each time the plot refreshes, the balls are getting closer to the viewer. This is reflected with the increase in size of the points as the animation progresses. Obviously, some pitches travel faster than others, which explains the different sizes within a particular frame. Animations revert to the initial point of release once *all* of the baseballs have reached home plate. During an interactive session, animateFX produces a series of plots that may not viewed easily. One option available to the user is to wrap `animation::saveHTML` around animateFX to view the animation in a browser with proper looping controls (Xie, 2013a).

To reduce the time and thinking required to produce these animations, animateFX has default settings for the geometry, color, opacity and size associated with each plot. Any of these assumptions can be altered - except for the point geometry. In order for animations to work, a data frame with the appropriately named `PITCHf/x` parameters (that is, `x0`, `y0`, `z0`, `vx0`, `vy0`, `vz0`, `ax0`, `ay0` and `az0`) is required. In Figure 6, every four-seam and cut fastball thrown by Rivera and Hughes during the 2011 season is visualized using the pitches data frame obtained earlier (the animation is available at <http://cpsievert.github.io/pitchRx/ani1>).

```
animateFX(pitches, layer=list(theme_bw(), coord_equal(),
  facet_grid(pitcher_name~stand, labeller = relabel)))
```

In the animation corresponding to Figure 6, the upper right-hand portion (Rivera throwing to right-handed batters) reveals the clearest pattern in flight trajectories. Around the point of release, Rivera's two pitch types are hard to distinguish. However, after a certain point, there is a very different flight path among the two pitch types. Specifically, the drastic left-to-right movement of the cut fastball is noticeably different from the slight right-to-left movement of the four-seam fastball. In recent years, cut fastballs have gained notoriety among the baseball community as a coveted pitch for pitchers have at their disposal. This is largely due to the difficulty that a batter has in distinguishing the cut fastball from another fastball as the ball travels toward home plate. Clearly, this presents an advantage for the pitcher since they can use deception to reduce batter's ability to predict where the ball will cross home

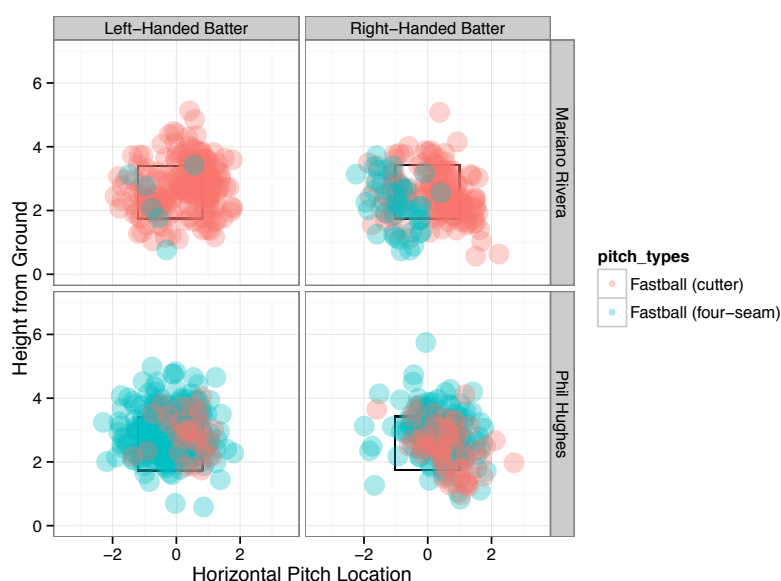


Figure 6: The last frame of an animation of every four-seam and cutting fastballs thrown by NY Yankee pitchers Mariano Rivera and Phil Hughes during the 2011 season. The actual animation can be viewed [here](#). Pitches are faceted by pitcher and batting stance. For instance, the top left plot portrays pitches thrown by Rivera to left-handed batters.

plate. This deception factor combined with Rivera's ability to locate his pitches explain his accolades as one of the greatest pitchers of all time (Traub, 2010).

Although we see a clear pattern in Rivera's pitches, MLB pitchers are hardly ever that predictable. Animating that many pitches for another pitcher can produce a very cluttered graphic which is hard to interpret (especially when many pitch types are considered). However, we may still want to obtain an indication of pitch trajectory over a set of many pitches. A way to achieve this is to average over the `PITCHf/x` parameters to produce an overall sense of pitch type behavior (via the `avg.by` option). Note that the facet variables are automatically considered indexing variables. That is, in Figure 7, there are eight 'average' pitches since there are two pitch types, two pitchers, and two types of batting stance (the animation is available at <http://cpsievert.github.io/pitchRx/ani2>).

```
animateFX(pitches, avg.by = "pitch_types", layer = list(coord_equal(), theme_bw(),
  facet_grid(pitcher_name~stand, labeller = relabel)))
```

Interactive 3D graphics

rgl is an R package that utilizes OpenGL for graphics rendering. `interactiveFX` utilizes **rgl** functionality to reproduce flight paths on an interactive 3D platform. Figure 8 has two static pictures of Mariano Rivera's 2011 fastballs on this interactive platform. This is great for gaining new perspectives on a certain set of pitches, since the trajectories can be viewed from any angle. Figure 8 showcases the difference in trajectory between Rivera's pitch types.

```
Rivera <- subset(pitches, pitcher_name == "Mariano Rivera")
interactiveFX(Rivera, avg.by = "pitch_types")
```

Conclusion

pitchRx utilizes **XML2R**'s convenient framework for manipulating XML content in order to provide easy access to `PITCHf/x` and related Gameday data. **pitchRx** removes access barriers which allows the average R user and baseball fan to spend their valuable time analyzing Gameday's enormous source of baseball information. **pitchRx** also provides a suite of functions that greatly reduce the amount of work involved to create popular `PITCHf/x` graphics. For those interested in obtaining other XML data, **pitchRx** serves as a nice example of leveraging **XML2R** to quickly assemble custom XML data collection mechanisms.

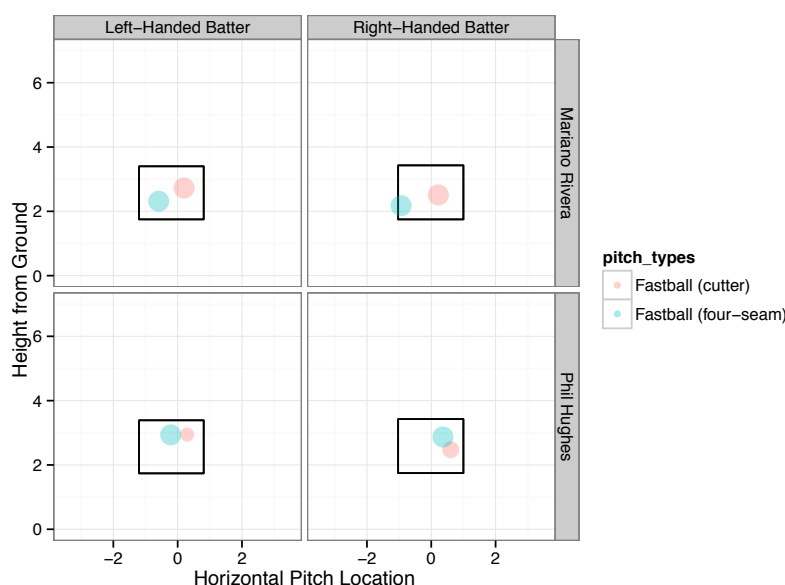


Figure 7: The last frame of an animation of ‘averaged’ four-seam and cutting fastballs thrown by NY Yankee pitchers Mariano Rivera and Phil Hughes during the 2011 season. The actual animation can be viewed [here](#). PITCHf/x parameters are averaged over pitch type, pitcher and batting stance. For instance, the bottom right plot portrays an “average four-seam” and “average cutter” thrown by Hughes to right-handed batters.

Acknowledgements

Many thanks to my major professor, Dr. Heike Hofmann, for her direction and support throughout this project. Thanks also to the anonymous reviewers for helpful feedback. This document was created using the [knitr](#) package [Xie \(2013b\)](#). The source files can be found [here](#).

Bibliography

- D. Adler, D. Murdoch, and others. *rgl: 3D Visualization Device System (OpenGL)*. URL <http://rgl.neoscientists.org>. R package version 0.93.963. [p6]
- A. Alt and M. S. White. Tracking an object with multiple asynchronous cameras, 09 2008. URL http://www.patentlens.net/patentlens/patent/US_7062320/. [p5]
- N. DiMeo. Baseball’s particle accelerator, Aug. 2007. URL http://www.slate.com/articles/sports/sports_nut/2007/08/baseballs_particle_accelerator.html. Last visited on 07/12/2012. [p5]
- M. Fast. How to build a pitch database, Aug. 2007. URL <http://fastballs.wordpress.com/2007/08/23/how-to-build-a-pitch-database/>. Last visited on 12/15/2012. [p6]
- M. Fast. Spinning yarn: A zone of their own, July 2011. URL <http://www.baseballprospectus.com/article.php?articleid=14572>. Last visited on 06/17/2013. [p13]
- E. Green and D. P. Daniels. What does it take to call a strike? three biases in umpire decision making. In *MIT Sloan Sports Analytics Conference*, 2014. [p12]
- M. Marchi and J. Albert. *Analyzing Baseball Data with R*. Chapman and Hall/CRC, 2013. ISBN 9781466570221. URL <http://baseballwithr.wordpress.com/>. [p6]
- B. Mills. Rethinking ‘loess’ for binomial-response PITCHf/x strike zone maps, Dec. 2010. URL <http://princeofslides.blogspot.com/2010/12/rethinking-loess-for-binomial-response.html>. Last visited on 06/20/2013. [p14]
- A. Nathan. A statistical study of PITCHf/x pitched baseball trajectories, Feb. 2008. URL <http://webusers.npl.illinois.edu/~a-nathan/pob/MCAalysis.pdf>. Last visited on 07/25/2012. [p5]

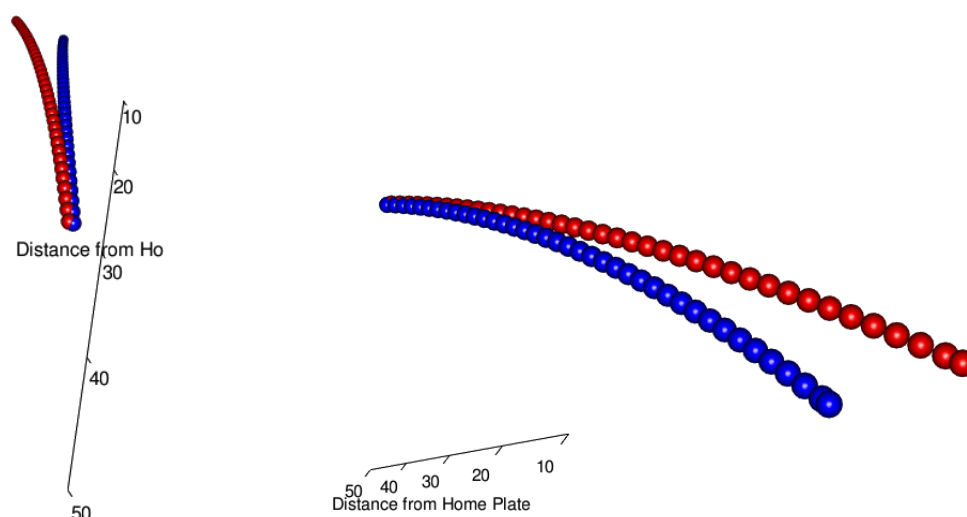


Figure 8: 3D scatterplot of pitches from Rivera. Pitches are plotted every one-hundredth of a second. Cutting fastballs are shown in red and four-seam fastballs are shown in blue. The left hand plot takes a viewpoint of Rivera and the right hand plot takes a viewpoint near the umpire. Note these are static pictures of an interactive object.

M. A. Pane, S. L. Ventura, R. C. Steorts, and A. C. Thomas. Trouble With The Curve: Improving MLB Pitch Classification. *ArXiv e-prints*, Apr. 2013. [p5]

R Special Interest Group on Databases. *DBI: R Database Interface*, 2013. URL <http://CRAN.R-project.org/package=DBI>. R package version 0.2-7. [p11]

C. Sievert. *XML2R: EasieR XML Data Collection*, 2014a. URL <http://cpsievert.github.com/XML2R>. R package version 0.0.4. [p6]

C. Sievert. *pitchRx: Tools for Scraping XML Data and Visualizing Major League Baseball PITCHfx*, 2014b. URL <http://cpsievert.github.com/pitchRx>. R package version 1.5. [p5]

J. Traub. Mariano Rivera, king of the closers, June 2010. URL <http://www.nytimes.com/2010/07/04/magazine/04Rivera-t.html>. Last visited on 02/04/2013. [p17]

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>. [p6]

H. Wickham and R. Francois. *dplyr: A Grammar of Data Manipulation*, 2014. R package version 0.2. [p11]

S. Wood. *Generalized Additive Models: An Introduction with R*. Chapman and Hall/CRC, 2006. [p14]

Y. Xie. animation: An R package for creating animations and demonstrating statistical methods. *Journal of Statistical Software*, 53(1):1–27, 2013a. URL <http://www.jstatsoft.org/v53/i01/>. [p16]

Y. Xie. *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, 2013b. ISBN 978-1482203530. [p18]

Carson Sievert
Department of Statistics
Iowa State University
sievert@iastate.edu

A Multiscale Test of Spatial Stationarity for Textured Images in R

by Matthew A. Nunes, Sarah L. Taylor and Idris A. Eckley

Abstract The ability to automatically identify areas of homogeneous texture present within a greyscale image is an important feature of image processing algorithms. This article describes the R package **LS2Wstat** which employs a recent wavelet-based test of stationarity for locally stationary random fields to assess such spatial homogeneity. By embedding this test within a quadtree image segmentation procedure we are also able to identify texture regions within an image.

Introduction

This paper provides an introduction to the **LS2Wstat** package (Taylor and Nunes, 2014), developed to implement recent statistical methodology for the analysis of (greyscale) textured images. Texture analysis is a branch of image processing concerned with studying the variation in an image surface; this variation describes the physical properties of an object of interest. The key applications in this field, namely discrimination, classification and segmentation, are often dependent on assumptions relating to the second-order structure (variance properties) of an image. In particular many techniques commonly assume that images possess the property of spatial stationarity (Gonzalez and Woods, 2001). However, for images arising in practice this assumption is often not realistic, i.e. typically the second-order structure of an image varies across location. It is thus important to test this assumption of stationarity before performing further image analysis. See Figure 1 for examples of textured images. For a comprehensive introduction to texture analysis, see Bishop and Nasrabadi (2006) or Petrou and Sevilla (2006).

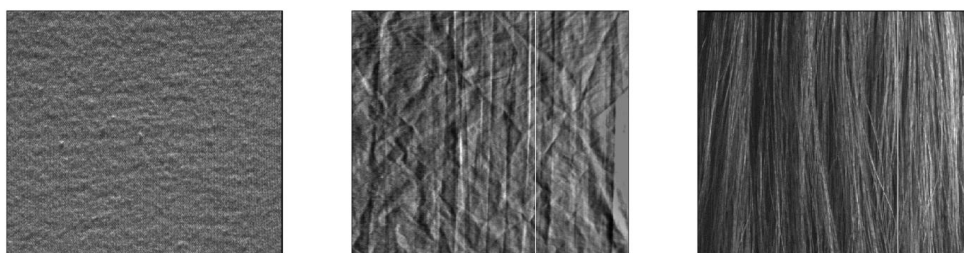


Figure 1: Examples of textured images: fabric, creased material and hair (available from the R package **LS2W**; Eckley and Nason, 2013).

Recently, Taylor et al. (in press) proposed a test of spatial stationarity founded on the *locally stationary two-dimensional wavelet* (LS2W) modelling approach of Eckley et al. (2010). The LS2W modelling approach provides a location-based decomposition of the spectral structure of an image. The $\text{Bootstat}_{\text{LS2W}}$ test proposed by Taylor et al. (in press) uses a statistic based on an estimate of the process variance within a hypothesis testing framework, employing bootstrap resampling under the null hypothesis assumption of stationarity to assess its significance.

Given a test of spatial stationarity for random fields, it is natural to consider how this might be usefully applied within a problem such as texture segmentation. The ability to determine non-stationarity and the presence of localised textured regions within images is important in a broad range of scientific and industrial applications, including product evaluation or quality control purposes. Possible areas of use for the methods described in this article include identifying uneven wear in fabrics (Chan and Pang, 2000; Taylor et al., in press) and defect detection on the surface of industrial components (Wilttschi et al., 2000; Bradley and Wong, 2001) or natural products (Funck et al., 2003; Pölzleitner, 2003). For a review of texture segmentation, see Pal and Pal (1993).

Readily available implementations for stationarity assessment have, up until now, been restricted to the time series setting; examples of such software in this context include the R add-on packages **urca** (Pfaff, 2008; Pfaff and Stigler, 2013), **CADFtest** (Lupi, 2009) and **locits** (Nason, 2013a,b).

Below we describe the package **LS2Wstat** which implements the spatial test of stationarity proposed by Taylor et al. (in press). The package has been developed in R and makes use of several functions within the **LS2W** package (Eckley and Nason, 2011, 2013). The article is structured as

follows. We begin by describing details of simulation of LS2W and other processes. An overview of the $\text{Bootstat}_{\text{LS2W}}$ test of stationarity is then given, focussing in particular on the function TOS2D . We then illustrate the application of the test on both simulated and real texture images. Finally, the article concludes by describing how the algorithm might be embedded within a quadtree image splitting procedure to identify regions of texture homogeneity within a multi-textured image.

Simulating textured images with LS2Wstat

Before describing the implementation of the work proposed in [Taylor et al. \(in press\)](#), we first explain how to generate candidate textured images using the **LS2Wstat** package. Several different spatially stationary and non-stationary random fields can be generated with the `simTexture` function. See the package help file for full details of the processes available.

To demonstrate the **LS2Wstat** implementation, throughout this article we consider a realisation of a white noise process with a subregion of random Normal deviates in its centre with a standard deviation of 1.6. This simulated texture type is called NS5, and is one of several textures which can be simulated from the package. In particular, we consider an image of dimension 512×512 with a central region that is a quarter of the image, i.e. a dimension size of 128×128 . This can be obtained as follows:

```
> library("LS2Wstat")
> set.seed(1)
> X <- simTexture(n = 512, K = 1, imtype = "NS5", sd = 1.6, prop = 0.25)[[1]]
> image(plotmtx(X), col = grey(255:0/256))
```

The `simTexture` function returns a list of length K with each list entry being a matrix representing an image of dimension $n \times n$ with the chosen spectral structure. In this case, since $K = 1$, a list of length 1 is returned. The simulated image X is shown in Figure 2. Note in particular that visually, one can discern that the image consists of two subtly different texture types. Firstly, the centre of the image has one particular form of second order structure. The second texture structure can be seen in the remainder of the image. Throughout the rest of this article we shall apply the approach of [Taylor et al. \(in press\)](#) to this image.

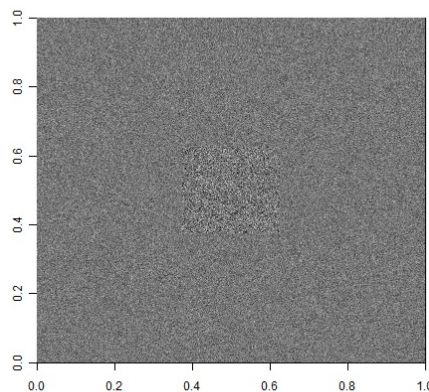


Figure 2: An example of a textured image (NS5) simulated with the `simTexture` function.

Testing the spatial stationarity of images

We now briefly introduce the LS2W random field model of [Eckley et al. \(2010\)](#) together with some associated notation, before describing the implementation of the test of stationarity proposed in [Taylor et al. \(in press\)](#). For an accessible introduction to wavelets, please see [Prasad and Iyengar \(1997\)](#), [Vidakovic \(1999\)](#) or [Nason \(2008\)](#).

The LS2W process model is defined by

$$X_r = \sum_l \sum_{j=1}^{\infty} \sum_u w_{j,u}^l \psi_{j,u}^l(r) \zeta_{j,u}^l, \quad (1)$$

for directions $l = h, v$ or d and spatial locations r , where $\{\zeta_{j,u}^l\}$ is a zero-mean random orthonormal increment sequence; $\{\psi_{j,u}^l\}$ is a set of discrete nondecimated wavelets and $\{w_{j,u}^l\}$ is a collection of

amplitudes, constrained to vary slowly over locations of an image (Eckley et al., 2010). In the above definition, we assume the image is of dyadic dimension, i.e. we have $\mathbf{r} = (r, s)$ with $r, s \in \{1, \dots, 2^J\}$ and where J is the coarsest observed scale.

Eckley et al. (2010) also define the *local wavelet spectrum* (LWS) associated with an LS2W process. The LWS for a given location $\mathbf{z} = \left(\frac{r}{2^J}, \frac{s}{2^J}\right) \in (0, 1)^2$, at scale j in direction l is $S_j^l(\mathbf{z}) \approx w_j^l(\mathbf{u}/\mathbf{R})^2$. The LWS provides a decomposition of the process variance at (rescaled) locations \mathbf{z} , directions l , and wavelet scales j . In practice the LWS is usually unknown and so needs to be estimated (see Eckley et al., 2010, for details). Spectral estimation using the LS2W model is implemented in R in the add-on package **LS2W** (Eckley and Nason, 2013). The **LS2Wstat** routines described below thus have a dependence on some functions from the **LS2W** package.

A test of stationarity for LS2W processes

Next we turn to describe the implementation of a test of stationarity within the **LS2Wstat** package. We focus on describing the `BootstatLS2W` approach implemented in the **LS2Wstat** package, referring the interested reader to Taylor et al. (in press) for details of other tests which might be employed. Throughout this section let us assume that we have some image $X_{\mathbf{r}}$ (as in Figure 2), whose second-order structure we wish to test for spatial stationarity. We assume that X is an LS2W process with associated unknown spectrum, S_j^ℓ for $j = 1, \dots, J$ and $\ell = v, h$ or d . Since the model in (1) assumes the process has zero mean, if necessary the image can be detrended. This can be done in R, for example, by using the core **stats** package function `medpolish`, which implements Tukey's median polish technique (Tukey, 1977).

Under the null hypothesis of stationarity, the wavelet spectrum will be constant across location for each scale and direction. Motivated by this fact Taylor et al. (in press) formulate a hypothesis test for the stationarity of the image $X_{\mathbf{r}}$ with

$$\begin{aligned} H_0 : S_j^\ell(\mathbf{z}) \text{ is constant across } \mathbf{z} \text{ for all } j \text{ and } \ell, \\ H_A : S_j^\ell(\mathbf{z}) \text{ is not constant across } \mathbf{z} \text{ for some } j \text{ or } \ell. \end{aligned}$$

Hence, a test statistic for the hypothesis should measure how much the wavelet spectrum for an observed image differs from constancy. Taylor et al. (in press) suggest using the average scale-direction spectral variance as a test statistic to measure the degree of non-stationary within an image, where the variance is taken over pixel locations, that is:

$$T \left\{ \hat{S}_j^\ell(\mathbf{z}) \right\} = \frac{1}{3J} \sum_{\ell} \sum_{j=1}^J \text{var}_{\mathbf{u}} \left(\hat{S}_{j,\mathbf{u}}^\ell \right). \quad (2)$$

In practice this statistic is computed based on an (unbiased) estimate of the local wavelet spectrum, produced by the **LS2W** function `cddews` (see the documentation in **LS2W** for details on optional arguments to this function). For the (square) image X , the test statistic is calculated using the function `avespecvar` as follows:

```
> TSvalue <- avespecvar(cddews(X, smooth = FALSE))
> TSvalue
[1] 0.2044345
```

Since the spectrum characterises the second-order structure of the observed random field (and hence its stationarity properties), Taylor et al. (in press) suggest determining the p-value of the hypothesis test by performing parametric bootstrapping. This corresponds to sampling LS2W processes assuming stationarity under the null hypothesis, and comparing the observed test statistic to that of the simulated LS2W processes under stationarity. For pseudo-code of this algorithm, please see Algorithm 1.

This bootstrap algorithm is performed with the **LS2Wstat** function `TOS2D`. The function has arguments:

`image`: The image you want to analyse.

`detrend`: A binary value indicating whether the image should be detrended before applying the bootstrap test. If set to `TRUE`, the image is detrended using Tukey's median polish method.

`nsamples`: The number of bootstrap simulations to carry out. This is the value B in the pseudocode given above. By default this takes the value 100.

`theTS`: This specifies the test statistic function to be used within the testing procedure to measure non-stationarity. The test statistic should be based on the local wavelet spectrum and by default is the function `avespecvar` representing the statistic (2).

Bootstat_{LS2W}:

1. Compute the estimate of the LWS for the observed image, $\hat{S}_j^l(z)$.
 2. Evaluate T (Equation (2)) on the observed image, call this value T^{obs} .
 3. Compute the pixel average stationary spectrum \tilde{S}_j^l by taking the average of spectrum values for each scale and direction.
 4. **Iterate** for i in 1 to B bootstraps:
 - (a) Simulate $X_r^{(i)}$ from the stationary LS2W model using squared amplitudes given by \tilde{S}_j^l and Gaussian process innovations.
 - (b) Compute the test statistic T on the simulated realisation, call this value $T^{(i)}$.
 5. Compute the p-value for the test as $p = \frac{1 + \#\{T^{obs} \leq T^{(i)}\}}{B+1}$.
-

Algorithm 1: The bootstrap algorithm for testing the stationarity of locally stationary images.

verbose: A binary value indicating whether informative messages should be printed.

...: Any optional arguments to be passed to the **LS2W** function `cddews`. See the documentation for the `cddews` function for more details.

Note that **TOS2D** uses the **LS2W** process simulation function `LS2Wsim` from the **LS2W** R package to simulate bootstrap realizations under the null hypothesis. The output of **TOS2D** is a list object of class "TOS2D", which describes pertinent quantities associated with the bootstrap test of stationarity. The object contains the following components:

`data.name`: The name of the image tested for stationarity.

`samples`: A vector of length `nsamples + 1` containing each of the test statistics calculated in the bootstrap test. The first element of the vector is the value of the test statistic calculated for the original image itself.

`statistic`: The statistic used in the test.

`p.value`: The bootstrap p-value associated with the test.

In particular, the object returns the measure of spectral constancy in the entry `statistic`, together with the p-value associated with the stationarity test (in the `p.value` component).

An example of the function call is

```
> Xbstest <- TOS2D(X, nsamples = 100)
```

Note that the p-value returned within the "TOS2D" object is computed using the utility function `getpval`, which returns the parametric bootstrap p-value for the test from the bootstrap test statistics provided by counting those test statistic values less than T^{obs} (see Davison et al., 1999, for more details). In other words, the p.value component is obtained by the following call:

```
> pval <- getpval(Xbstest$samples)
Observed bootstrap is 0.204
p-value is 0.00990099
```

This p-value can then be used to assess the stationarity of a textured image region.

Information on the "TOS2D" class object can be obtained using the `print` or `summary` S3 methods for this class. For example, using the `summary` method, one would obtain

```
> summary(Xbstest)

2D bootstrap test of stationarity
object of class TOS2D
-----

summary
=====
data: X
Observed test statistic: 0.204
bootstrap p-value: 0.01
```

Alternatively, the print method for the "TOS2D" class prints more information about the Xbtest object. Note that the function internally calls the summary method for "TOS2D" objects:

```
2D bootstrap test of stationarity
  object of class TOS2D
-----

summary
=====
data: X
Observed test statistic: 0.204
bootstrap p-value: 0.01

Number of bootstrap realizations: 100
spectral statistic used: avespecvar
```

Other textured images

To demonstrate the test of stationarity further, we now provide some other textured image examples. Firstly, we consider a *Haar wavelet random field* with a diagonal texture, an example of a LS2W process as described in [Eckley et al. \(2010\)](#). The realisation of the process (shown in Figure 2) is simulated using the `simTexture` function with the command:

```
> Haarimage <- simTexture(512, K = 1, imtype = "S5")[[1]]
```

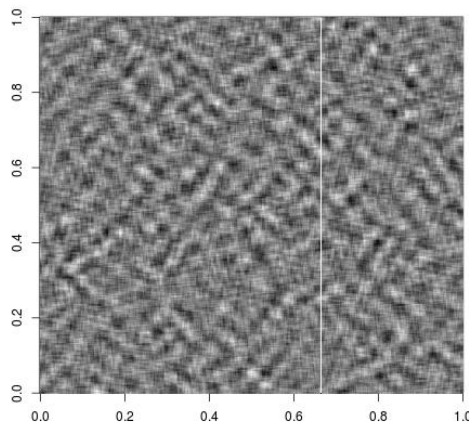


Figure 2: A realisation of a stationary LS2W process, Haar image, with a diagonal texture.

The test of stationarity of [Taylor et al. \(in press\)](#) performed on the image `Haarimage` with the function `TOS2D` reveals that the image is spatially stationary as expected, with a high p-value associated to the test.

```
> Haarimtest <- TOS2D(Haarimage, smooth = FALSE, nsamples = 100)
> summary(Haarimtest)
```

```
2D bootstrap test of stationarity
  object of class TOS2D
-----

summary
=====
data: Haarimage
Observed test statistic: 0.631
bootstrap p-value: 0.673

Number of bootstrap realizations: 100
spectral statistic used: avespecvar
```

As another example of a textured image, we construct an image montage using two of the textures shown in Figure 1 from the package **LS2W**. The montage, `montage1`, is shown in Figure 3.

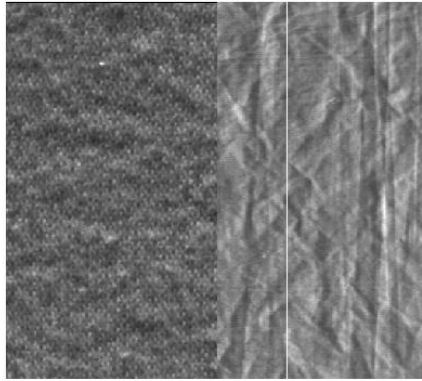


Figure 3: An example of an image montage, `montage1`, using two of the textures from Figure 1.

Note that since this image may not have zero mean as assumed by the LS2W model (1), we detrend the montage first using the `medpolish` function in the **stats** package.

```
> data(textures)
> montage1 <- cbind(A[1:512, 1:256], B[, 1:256])
> montage1zm <- medpolish(montage1)$residuals
```

The TOS2D test indicates that the texture montage is non-stationary:

```
> montage1zmtest <- TOS2D(montage1zm, smooth = FALSE, nsamples = 100)
> summary(montage1zmtest)
```

```
2D bootstrap test of stationarity
  object of class TOS2D
-----
```

```
summary
=====
data: montage1zm
Observed test statistic: 0
bootstrap p-value: 0.01
```

```
Number of bootstrap realizations: 100
spectral statistic used: avespecvar
```

Identifying areas of homogeneous texture using the bootstrap test of stationarity

In this section we describe embedding a test of stationarity into a quadtree algorithm to identify regions of spatial homogeneity within a textured image. This segmentation approach is similar in spirit to, e.g., [Spann and Wilson \(1985\)](#) or [Pal and Pal \(1987\)](#) which use homogeneity measures within a quadtree structure. We first give details of the quadtree implementation, and subsequently describe functions to aid illustration of quadtree decompositions.

A quadtree algorithm implementation

In essence, a region splitting algorithm recursively subdivides an input image into smaller regions, with the subdivision decisions being based on some statistical criterion. More specifically, in a *quadtree representation*, at each stage a (sub)image is divided into its four subquadrants if the criterion is not satisfied (see e.g., [Sonka et al., 1999](#)). The statistical criterion we use is spatial homogeneity, that is, a quadrant is further divided if it is considered as non-stationary by the $\text{Bootstat}_{\text{LS2W}}$ test. In practice, the quadtree implementation in **LS2Wstat** continues until all subregions are considered as stationary, or until the subregions reach a particular minimal dimension. The motivation for this is to ensure that we obtain statistically meaningful decisions using the stationarity test by not allowing too small a

Quadtree decomposition:

For an input image X :

Use the `BootstatLS2W` test to assess whether X is second-order stationary. If stationary, stop. If not,

1. Divide the image into four quadrants.
 2. For each quadrant, assess its stationarity with the `BootstatLS2W` test.
 3. For each quadrant assessed as non-stationary, recursively repeat steps 1–2, until the minimum testing region is reached or until all sub-images are judged to be stationary.
-

Algorithm 2: The quadtree algorithm for segmenting an image into regions of spatial stationarity.

testing sub-image. This procedure segments an image into regions of spatial stationarity. The quadtree algorithm is summarised in Algorithm 2.

Each image is further split if deemed non-stationary, which is determined by a test of stationarity such as `TOS2D`. After the first subdivision of an image, each sub-image is of size $n/2 \times n/2$. The sizes of the regions halve in size at each progressive division but increase in number. The R function in `LS2Wstat` which creates the quadtree structure described in Algorithm 2 is `imageQT`. The function has inputs:

`image`: The image to be decomposed with the quadtree algorithm.

`test`: A function for assessing regions of spatial homogeneity, for example `TOS2D`.

`minsize`: The testing size of sub-images below which we should not apply the function test.

`alpha`: The significance level of the `BootstatLS2W` test, with which to assess spatial stationarity of textured regions.

`...`: Any other optional arguments to test.

As an illustration of using the `imageQT` function, consider the code below to decompose the (non-stationary) input image X . We use the function `TOS2D` to assess the regions of spatial homogeneity although the `imageQT` function allows other functions to be supplied.

```
> QTdecX <- imageQT(X, test = TOS2D, nsamples = 100)
```

The output of the `imageQT` function is a list object of class "imageQT" with components:

`ind1`: The index representation of the non-stationary images in the quadtree decomposition.

`res1`: The results of the stationarity testing (from the `test` argument) during the quadtree decomposition. The results giving FALSE correspond to those non-stationary sub-images contained in the `ind1` component and the results giving TRUE correspond to the stationary sub-images, i.e. those contained in the `indS` component.

`indS`: The index representation of the stationary images in the quadtree decomposition.

This particular way of splitting an image has a convenient indexing representation to identify the position of subregions within an image. If a (sub)image is subdivided into quadrants, we assign it a base 4 label as follows: 0 – top-left quadrant; 1 – bottom-left quadrant; 2 – top-right quadrant; 3 – bottom-right quadrant. By continuing in this manner, we can assign an index to each tested subregion, with the number of digits in the index indicating how many times its parent images have been subdivided from the "root" of the tree (the original image). This indexing is illustrated for the quadtree decomposition given in the example in Figure 3.

Examining the quadtree decomposition of the image X using the `print` S3 method for the "imageQT" class, we have

```
> print(QTdecX)
```

```
2D quadtree decomposition
object of class imageQT
-----
```

```
summary
```

```
=====
```

```
data: X
```

00	02		20		22
01	030	032	210	212	23
	031	033	211	213	
10	120	122	300	302	32
	121	123	301	303	
11	13		31		33

Figure 3: An example of a quadtree decomposition. The location of the sub-images in the decomposition are described by the indexing system described in the text.

Indices of non-stationary sub-images:

"0" "1" "2" "3" "03" "12" "21" "30"

Indices of stationary sub-images:

"00" "01" "02" "10" "11" "13" "20" "22" "23" "31" "32" "33" "030" "031" "032" "033"
"120" "121" "122" "123" "210" "211" "212" "213" "300" "301" "302" "303"

minimum testing region: 64

The `res1` component gives the results of the test of stationarity for all sub-images tested during the quadtree procedure, reporting FALSE for the non-stationary sub-images and TRUE for the stationary ones:

```
> QTdecX$res1
[1] FALSE
[[2]]
[1] FALSE FALSE FALSE FALSE
[[3]]
[1] TRUE TRUE TRUE FALSE TRUE TRUE FALSE TRUE TRUE FALSE TRUE TRUE
[13] FALSE TRUE TRUE TRUE
[[4]]
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[16] TRUE
```

Plotting a quadtree decomposition

By performing the quadtree algorithm given in Algorithm 2, it is possible to decompose images into regions indicating regional stationarity. Note that if a texture discrimination procedure is used to classify the output from the stationarity quadtree algorithm, the image segmentation method can be seen as a *split and merge* technique.

Suppose we have performed the quadtree decomposition. The **LS2Wstat** package includes an `S3` plot method for "imageQT" objects to plot the output for the "imageQT" class and optionally a classification of those textured regions. If the classification output is plotted (`class = TRUE`), each textured region is uniquely coloured according to its texture group. The function has arguments:

`x`: A quadtree decomposition object, such as output from `imageQT`.

`cires`: Vector of class labels associated to the subimages produced by the quadtree decomposition.

`unclassval`: A value for any unclassified values in a quadtree decomposition.

`class`: A Boolean value indicating whether to plot the classification of the quadtree subimages.

`QT`: A Boolean value indicating whether to plot the quadtree decomposition.

We now illustrate the use of this function with the example given in Figure 2. Suppose the textured regions identified by the quadtree algorithm in the `QTdecX` object have been classified according to some texture discrimination procedure. For the purposes of this example, we suppose that the 28 regions of stationarity in `QTdecX` (see Figure 3) have been classified as coming from two groups according to the labels

```
> texclass <- c(rep(1, times = 15), rep(c(2, 1, 1), times = 4), 1)
> texclass
[1] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 1 1 2 1 1 2 1 1 1
```

Using the output from the quadtree technique (QTdecX) and the texture classification vector `texclass`, we can use the quadtree plotting function for "imageQT" objects as follows:

```
> plot(QTdecX, texclass, class = TRUE, QT = TRUE)
> plot(QTdecX, texclass, class = TRUE, QT = FALSE)
```

The quadtree decomposition from this example is shown in Figure 4a; the same decomposition is shown together with the texture classification in Figure 4b.

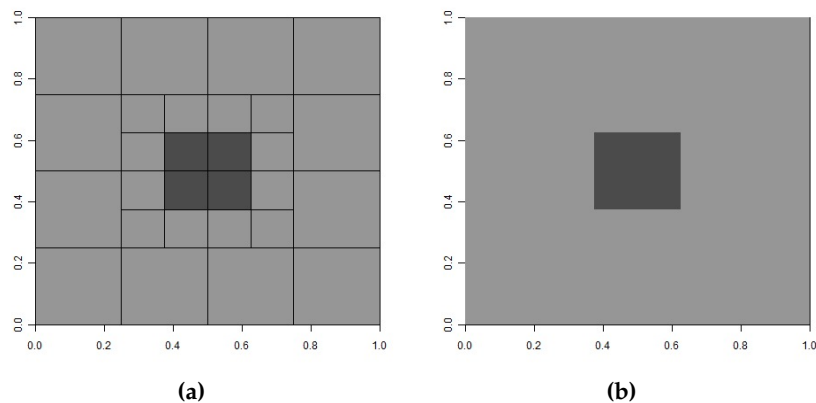


Figure 4: An example of a quad-tree decomposition using `imageQT`, together with an assumed sub-image texture classification.

We also consider an image montage using the textures from the package **LS2W**. The montage `Y` is shown in Figure 5. Prior to performing the quadtree decomposition, we detrend the image.

```
> data(textures)
> Y <- cbind(A[1:512, 1:256], rbind(B[1:256, 1:256], C[1:256, 1:256]))
> Yzm <- medpolish(Y)$residuals
```

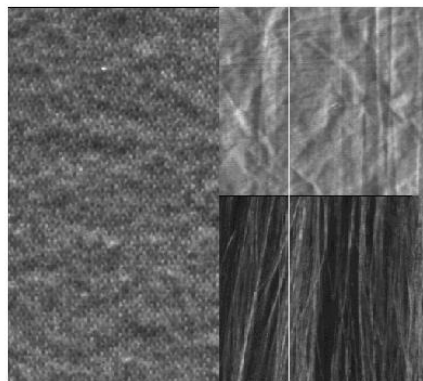


Figure 5: An example of an image montage, `Y`, using the textures from Figure 1.

Similarly to above, we can now perform a quadtree decomposition of the image `Y`:

```
> QTdecYzm <- imageQT(Yzm, test = TOS2D, nsamples = 100)
> print(QTdecYzm)
```

```
2D quadtree decomposition
object of class imageQT
```

```
-----
```

```
summary
=====
data: Yzm
Indices of non-stationary sub-images:

Indices of stationary sub-images:
"0" "1" "2" "3"

minimum testing region: 64
```

The function `imageQT` initially assesses that the image is indeed non-stationary, and then proceeds to analyse sub-images of the montage. The algorithm stops the quadtree decomposition after the first decomposition level, since it judges all quadrants of the image to be stationary, described by the indices "0", "1", "2", and "3".

Summary

In this article we have described the **LS2Wstat** package, which implements some recent methodology for image stationarity testing (Taylor et al., *in press*). Our algorithm is most useful as a test of homogeneity in textures which are visually difficult to assess. We have also extended its potential use by embedding it within a quadtree implementation, allowing assessment of potentially multi-textured images. The implementation is demonstrated using simulated and real textures throughout the paper.

Acknowledgements

We thank Aimée Gott for suggestions on an early version of the package. We would also like to thank Heather Turner, two anonymous referees and the Editor for helpful comments which have resulted in an improved manuscript and package.

Bibliography

- C. M. Bishop and N. M. Nasrabadi. *Pattern Recognition and Machine Learning*, volume 1. Springer-Verlag, New York, 2006. [p20]
- C. Bradley and Y. S. Wong. Surface texture indicators of tool wear – a machine vision approach. *The International Journal of Advanced Manufacturing Technology*, 17(6):435–443, 2001. [p20]
- C. Chan and G. K. H. Pang. Fabric defect detection by Fourier analysis. *IEEE Transactions on Industry Applications*, 36(5):1267–1276, 2000. [p20]
- A. Davison, D. Hinkley, and A. J. Canty. *Bootstrap Methods and their Application*. Cambridge University Press, 1999. [p23]
- I. A. Eckley and G. P. Nason. LS2W: Locally stationary wavelet fields in R. *Journal of Statistical Software*, 43(3):1–23, 2011. URL <http://www.jstatsoft.org/v43/i03>. [p20]
- I. A. Eckley and G. P. Nason. *LS2W: Locally Stationary Two-Dimensional Wavelet Process Estimation Scheme*, 2013. URL <http://CRAN.R-project.org/package=LS2W>. R package version 1.3-3. [p20, 22]
- I. A. Eckley, G. P. Nason, and R. L. Treloar. Locally stationary wavelet fields with application to the modelling and analysis of image texture. *Journal of the Royal Statistical Society C*, 59(4):595–616, 2010. [p20, 21, 22, 24]
- J. W. Funck, Y. Zhong, D. A. Butler, C. C. Brunner, and J. B. Forrer. Image segmentation algorithms applied to wood defect detection. *Computers and Electronics in Agriculture*, 41(1):157–179, 2003. [p20]
- R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Prentice Hall, 2nd edition, 2001. [p20]
- C. Lupi. Unit root CADF testing with R. *Journal of Statistical Software*, 32(2):1–19, 2009. URL <http://www.jstatsoft.org/v32/i02>. [p20]
- G. P. Nason. *Wavelet Methods in Statistics with R*. Springer-Verlag, 2008. [p21]
- G. P. Nason. *locits: Test of Stationarity and Localized Autocovariance*, 2013a. URL <http://CRAN.R-project.org/package=locits>. R package version 1.4. [p20]

- G. P. Nason. A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *Journal of the Royal Statistical Society B*, 75(5): 879–904, 2013b. [p20]
- N. R. Pal and S. K. Pal. A review on image segmentation techniques. *Pattern Recognition*, 26(9): 1277–1294, 1993. [p20]
- S. K. Pal and N. R. Pal. Segmentation using contrast and homogeneity measures. *Pattern Recognition*, 5 (4):293–304, 1987. [p25]
- M. Petrou and P. G. Sevilla. *Image Processing: Dealing with Texture*. John Wiley & Sons, 2006. [p20]
- B. Pfaff. *Analysis of Integrated and Cointegrated Time Series with R*. Springer-Verlag, New York, 2nd edition, 2008. [p20]
- B. Pfaff and M. Stigler. *urca: Unit Root and Cointegration Tests for Time Series Data*, 2013. URL <http://CRAN.R-project.org/package=urca>. R package version 1.2-8. [p20]
- W. Pölzleitner. Quality classification of wooden surfaces using Gabor filters and genetic feature optimisation. In *Machine Vision for the Inspection of Natural Products*, pages 259–277. Springer-Verlag, 2003. [p20]
- L. Prasad and S. S. Iyengar. *Wavelet Analysis with Applications to Image Processing*. CRC Press, 1997. [p21]
- M. Sonka, R. Boyle, and V. Hlavac. *Image Processing, Analysis, and Machine Vision*. PWS Publishing, 2nd edition, 1999. [p25]
- M. Spann and R. Wilson. A quad-tree approach to image segmentation which combines statistical and spatial information. *Pattern Recognition*, 18(3/4):257–269, 1985. [p25]
- S. Taylor and M. A. Nunes. *LS2Wstat: A Multiscale Test of Spatial Stationarity for LS2W Processes*, 2014. URL <http://CRAN.R-project.org/package=LS2Wstat>. R package version 2.0-3. [p20]
- S. L. Taylor, I. A. Eckley, and M. A. Nunes. A test of stationarity for textured images. *Technometrics*, in press. doi: 10.1080/00401706.2013.823890. [p20, 21, 22, 24, 29]
- J. Tukey. *Exploratory Data Analysis*. Addison-Wesley, 1977. [p22]
- B. Vidakovic. *Statistical Modelling by Wavelets*. John Wiley & Sons, New York, 1999. [p21]
- K. Wiltschi, A. Pinz, and T. Lindeberg. An automatic assessment scheme for steel quality inspection. *Machine Vision and Applications*, 12(3):113–128, 2000. [p20]

Matthew A. Nunes
Lancaster University
Lancaster
UK
m.nunes@lancaster.ac.uk

Sarah L. Taylor
Lancaster University
Lancaster
UK
tsarah8624@yahoo.co.uk

Idris A. Eckley
Lancaster University
Lancaster
UK
i.eckley@lancaster.ac.uk

Stratified Weibull Regression Model for Interval-Censored Data

by Xiangdong Gu, David Shapiro, Michael D. Hughes and Raji Balasubramanian

Abstract Interval censored outcomes arise when a silent event of interest is known to have occurred within a specific time period determined by the times of the last negative and first positive diagnostic tests. There is a rich literature on parametric and non-parametric approaches for the analysis of interval-censored outcomes. A commonly used strategy is to use a proportional hazards (PH) model with the baseline hazard function parameterized. The proportional hazards assumption can be relaxed in stratified models by allowing the baseline hazard function to vary across strata defined by a subset of explanatory variables. In this paper, we describe and implement a new R package **straweib**, for fitting a stratified Weibull model appropriate for interval censored outcomes. We illustrate the R package **straweib** by analyzing data from a longitudinal oral health study on the timing of the emergence of permanent teeth in 4430 children.

Introduction

In many clinical studies, the time to a silent event is known only up to an interval defined by the times of the last negative and first positive diagnostic test. Event times arising from such studies are referred to as ‘interval-censored’ data. For example, in pediatric HIV clinical studies, the timing of HIV infection is known only up to the interval from the last negative to the first positive HIV diagnostic test (Dunn et al., 2000). Examples of interval-censored outcomes can also be found in many other medical studies (Gomez et al., 2009).

A rich literature exists on the analysis of interval-censored outcomes. Non-parametric approaches include the self-consistency algorithm for the estimation of the survival function (Turnbull, 1976). A semi-parametric approach based on the proportional hazards model has been developed for interval-censored data (Finkelstein, 1986; Goetghebeur and Ryan, 2000). A variety of parametric models can also be used to estimate the distribution of the time to the event of interest, in the presence of interval-censoring (Lindsey and Ryan, 1998). An often used parametric approach for the analysis of interval-censored data is based on the assumption of a Weibull distribution for the event times (Lindsey and Ryan, 1998). The Weibull distribution is appropriate for modeling event times when the hazard function can be reliably assumed to be monotone. Covariate effects can be modeled through the assumption of proportional hazards (PH), which assumes that the ratio of hazard functions when comparing individuals in different strata defined by explanatory variables is time-invariant. The article by Gomez et al. (2009) presents a comprehensive review of the state-of-the-art techniques available for the analysis of interval-censored data.

In this paper, we implement a parametric approach for modeling covariates applicable to interval-censored outcomes, but where the assumption of proportional hazards may be questionable for a certain subset of explanatory variables. For this setting, we implement a stratified Weibull model by relaxing the PH assumption across levels of a subset of explanatory variables. We compare the proposed model to an alternative stratified Weibull regression model that is currently implemented in the R package **survival** (Therneau, 2012). We illustrate the difference between these two models analytically and through simulation.

The paper is organized as follows: In Section 2, we present and compare two models for relaxing the PH assumption, based on the assumption of a Weibull distribution for the time to event of interest. In this section, we discuss estimation of the unknown parameters of interest, hazard ratios comparing different groups of subjects based on specific values of explanatory covariates and tests of the PH assumption. These methods are implemented in a new R package, **straweib** (Gu and Balasubramanian, 2013). In Section 3, we perform simulation studies to compare two stratified Weibull models implemented in R packages **straweib** and **survival**. In Section 4, we illustrate the use of the R package **straweib** by analyzing data from a longitudinal oral health study on the timing of the emergence of permanent teeth in 4430 children in Belgium (Leroy et al., 2003; Gomez et al., 2009). In Section 5, we discuss the models implemented in this paper and present concluding remarks.

Weibull regression models

Let T denote the continuous, non-negative random variable corresponding to the time to event of interest, with corresponding probability distribution function (pdf) and cumulative distribution func-

tion (cdf), denoted by $f(t)$ and $F(t)$, respectively. We let $S(t) = 1 - F(t)$ to denote the corresponding survival function and $h(t) = \lim_{\delta t \rightarrow 0} \frac{P(t \leq T < t + \delta t | T \geq t)}{\delta t}$ to denote the hazard function. We let \mathbf{Z} denote the $p \times 1$ vector of explanatory variables or covariates.

We assume that the random variable $T | \mathbf{Z} = \mathbf{0}$ is distributed according to a Weibull distribution, with scale and shape parameters denoted by λ and γ , respectively. The well known PH model to accommodate the effect of covariates on T is expressed as:

$$h(t | \mathbf{Z}) = h(t | \mathbf{Z} = \mathbf{0}) \times \exp(\boldsymbol{\beta}' \mathbf{Z}),$$

where $\boldsymbol{\beta}$ denotes the $p \times 1$ vector of regression coefficients corresponding to the vector of explanatory variables, \mathbf{Z} .

Thus, under the Weibull PH model, the survival and hazard functions corresponding to T can be expressed as

$$S(t | \mathbf{Z}) = \exp(-\lambda \exp(\boldsymbol{\beta}' \mathbf{Z}) t^\gamma) \quad (1)$$

$$h(t | \mathbf{Z}) = \lambda \exp(\boldsymbol{\beta}' \mathbf{Z}) \gamma t^{\gamma-1} \quad (2)$$

where, $\lambda > 0$ and $\gamma > 0$ correspond to the scale and shape parameters corresponding to T when $\mathbf{Z} = \mathbf{0}$. The hazard ratio comparing two individuals with covariate vectors \mathbf{Z} and \mathbf{Z}^* is equal to $\exp(\boldsymbol{\beta}'(\mathbf{Z} - \mathbf{Z}^*))$.

Stratified Weibull regression model implemented in the R package survival

In this section, we describe the stratified Weibull PH regression model implemented in the the R package **survival** (Therneau, 2012).

Consider the following log-linear model for the random variable T :

$$\log(T | \mathbf{Z}) = \mu + \alpha_1 Z_1 + \cdots + \alpha_p Z_p + \sigma \epsilon$$

where, $\alpha_1, \dots, \alpha_p$ denote unknown regression coefficients corresponding to the p dimensional vector of explanatory variables, μ denotes the intercept, and σ denotes the scale parameter. The random variable ϵ captures the random deviation of event times on the natural logarithm scale (i. e. $\log(T)$) from the linear model as a function of the covariate vector \mathbf{Z} . In general, the log-linear form of the model for T can be shown to be equivalent to the accelerated failure time (AFT) model (Collett, 2003).

The assumption of a standard Gumbel distribution with location and scale parameters equal to 0 and 1, respectively, implies that the random variable T follows a Weibull distribution. Moreover, in this case, both the PH and AFT assumptions (or equivalently, the log-linear model) lead to identical models with different parameterizations (Collett, 2003). The survival and hazard functions can be expressed as:

$$S(t | \mathbf{Z}) = \exp \left[- \exp \left(\frac{\log(t) - \mu - \boldsymbol{\alpha}' \mathbf{Z}}{\sigma} \right) \right] \quad (3)$$

$$h(t | \mathbf{Z}) = \exp \left[- \frac{\mu + \boldsymbol{\alpha}' \mathbf{Z}}{\sigma} \right] \frac{1}{\sigma} t^{\frac{1}{\sigma}-1} \quad (4)$$

The coefficients for the explanatory variables ($\boldsymbol{\beta}$) in the hazard function ($h(t | \mathbf{Z})$) are equal to $-\frac{\boldsymbol{\alpha}}{\sigma}$. Moreover, there is a one-one correspondence between the parameters $\lambda, \gamma, \boldsymbol{\beta}$ in equations (1)-(2) and the parameters $\mu, \sigma, \boldsymbol{\alpha}$ in equations (3)-(4), where $\lambda = \exp(-\frac{\mu}{\sigma})$, $\gamma = \sigma^{-1}$ and $\boldsymbol{\beta}_j = -\frac{\alpha_j}{\sigma}$ (Collett, 2003).

The log-linear form of the Weibull model can be generalized to allow arbitrary baseline hazard functions within subgroups defined by a stratum indicator $S = 1, \dots, s$. Thus, the stratified Weibull regression model for an individual in the j^{th} stratum is expressed as:

$$\log(T | \mathbf{Z}, S = j) = \mu_j + \alpha_1 Z_1 + \cdots + \alpha_p Z_p + \sigma_j \epsilon$$

where μ_j and σ_j denote stratum specific intercept and scale parameters. This model is implemented in the R package **survival** (Therneau, 2012). In this model, the regression coefficients $\boldsymbol{\alpha}$ on the AFT scale are assumed to be stratum independent.

However, the hazard ratio comparing two individuals with covariate vectors and stratum indicators denoted by $(\mathbf{Z}, S = j)$ and $(\mathbf{Z}^*, S = k)$ is stratum specific and is given by:

$$\frac{h(t | S = j, \mathbf{Z})}{h(t | S = k, \mathbf{Z}^*)} = t^{1/\sigma_j - 1/\sigma_k} \frac{\sigma_k}{\sigma_j} \exp \left(\frac{\mu_k}{\sigma_k} - \frac{\mu_j}{\sigma_j} \right) \exp \left(\boldsymbol{\alpha}' \left(\mathbf{Z}^* / \sigma_k - \mathbf{Z} / \sigma_j \right) \right)$$

For $j \neq k$, the hazard ratio varies with time t . However, when $j = k$, the hazard ratio comparing two individuals within the same stratum $S = j$ is invariant with respect to time t but is stratum-dependent and reduces to:

$$\frac{h(t | S = j, \mathbf{Z})}{h(t | S = j, \mathbf{Z}^*)} = \exp \left(\frac{\alpha'}{\sigma_j} (\mathbf{Z}^* - \mathbf{Z}) \right) \quad (5)$$

Stratified Weibull regression model implemented in R package straweib

In this section, we describe the stratified Weibull regression model that is implemented in the new R package, **straweib** (Gu and Balasubramanian, 2013).

To relax the proportional hazards assumption in the Weibull regression model, we propose the following model for an individual in the stratum $S = j$:

$$h(t | \mathbf{Z}, S = j) = \lambda_j \exp(\beta' \mathbf{Z}) \gamma_j t^{\gamma_j - 1} \quad (6)$$

Equivalently, the model can be stated in terms of the survival function as:

$$S(t | \mathbf{Z}, S = j) = \exp \left(-\lambda_j \exp(\beta' \mathbf{Z}) t^{\gamma_j} \right)$$

Here, we assume that the scale and shape parameters (λ, γ) are stratum specific - however, the regression coefficients β are assumed to be constant across strata (S). The hazard ratio comparing two individuals with covariate vectors and stratum indicators denoted by $(\mathbf{Z}, S = j)$ and $(\mathbf{Z}^*, S = k)$ is given by:

$$\frac{h(t | S = j, \mathbf{Z})}{h(t | S = k, \mathbf{Z}^*)} = t^{\gamma_j - \gamma_k} \exp(\beta' (\mathbf{Z} - \mathbf{Z}^*)) \frac{\lambda_j \gamma_j}{\lambda_k \gamma_k}$$

For $j \neq k$, the hazard ratio varies with time t and thus relaxes the PH assumption. However, for $j = k$, the hazard ratio comparing two individuals within the same stratum $S = j$ reduces to:

$$\frac{h(t | S = j, \mathbf{Z})}{h(t | S = j, \mathbf{Z}^*)} = \exp(\beta' (\mathbf{Z} - \mathbf{Z}^*)) \quad (7)$$

This hazard ratio is invariant with respect to time t and stratum S , as in the stratified Cox model (Collett, 2003).

Estimation

Let $u_j = \log(\lambda_j)$ and $v_j = \log(\gamma_j)$. Let n_j denote the number of subjects in stratum $S = j$. For the k^{th} subject in stratum j , let \mathbf{Z}_{jk} denote the p dimensional vector of covariates and let a_{jk} and b_{jk} denote the left and right endpoints of the censoring interval. That is, a_{jk} denotes the time of the last negative test and b_{jk} denotes the time of the first positive test for the event of interest. Then the log-likelihood function can be expressed as:

$$l(\mathbf{v}, \mathbf{u}, \beta) = \sum_{j=1}^s \sum_{k=1}^{n_j} \log \{ \exp[-\exp(u_j + \beta' \mathbf{Z}_{jk} + \exp(v_j) \log(a_{jk}))] - \exp[-\exp(u_j + \beta' \mathbf{Z}_{jk} + \exp(v_j) \log(b_{jk}))] \}$$

The unknown parameters to be estimated are \mathbf{v} , \mathbf{u} , and β . The log-likelihood function can be optimized using the `optim` function in R. The shape and scale parameters can be estimated from the estimates of \mathbf{v} and \mathbf{u} . The covariance matrix of the estimates of these unknown parameters can be obtained by inverting the negative Hessian matrix that is output from the optimization routine (Cox and Hinkley, 1979).

Test of the PH assumption

One can test whether or not the baseline hazard functions of each strata are proportional to each other, by testing the equality of shape parameters across strata $S = 1, \dots, s$. That is,

$$H_0 : \gamma_1 = \gamma_2 = \dots = \gamma_s$$

or equivalently,

$$H_0 : v_1 = v_2 = \dots = v_s.$$

The null hypothesis H_0 can be tested using a likelihood ratio test, by comparing a reduced model that assumes that $\gamma_1 = \gamma_2 = \dots = \gamma_s$ to the full model in (6) assuming stratum specific shape parameters. We note that the reduced model is equivalent to the Weibull PH model that includes the stratum indicator S as an explanatory variable. Thus the reduced model has $s - 1$ fewer parameters than the

stratified model, or the full model. Let l_F and l_R denote the log-likelihoods of the full and reduced models evaluated at their MLE. Then the test statistic $T = -2(l_R - l_F)$ follows a χ^2_{s-1} distribution under H_0 . In addition to the likelihood ratio test, one can also use a Wald test to test the null hypothesis H_0 . The R package **straweib** illustrated in Section 3 outputs both the Wald and Likelihood Ratio test statistics.

Estimating hazard ratios

The log hazard ratio comparing two individuals with covariate vectors and stratum indicators denoted by $(\mathbf{Z}, S = j)$ and $(\mathbf{Z}^*, S = j^*)$ at time t can be expressed as:

$$r_{tjj^*} = \log(R_{tjj^*}) = u_j + v_j + \log(t) \exp(v_j) - u_{j^*} - v_{j^*} - \log(t) \exp(v_{j^*}) + \beta'(\mathbf{Z} - \mathbf{Z}^*)$$

Let \hat{v} , \hat{u} and $\hat{\beta}$ denote the maximum likelihood estimates for v , u and β , then r_{tjj^*} can be estimated by

$$\hat{r}_{tjj^*} = \hat{u}_j + \hat{v}_j + \log(t) \exp(\hat{v}_j) - \hat{u}_{j^*} - \hat{v}_{j^*} - \log(t) \exp(\hat{v}_{j^*}) + \hat{\beta}'(\mathbf{Z} - \mathbf{Z}^*)$$

Let $\mathbf{w} = (v, u, \beta) = (v_1, v_2, \dots, v_s, u_1, u_2, \dots, u_s, \beta_1, \dots, \beta_p)$. Let $\hat{\Sigma}$ denote the estimate of the covariance matrix of $\hat{\mathbf{w}}$. Let \mathbf{J}_{tjj^*} denote the Jacobian vector, $\mathbf{J}_{tjj^*} = \frac{\partial r_{tjj^*}}{\partial \mathbf{w}}|_{\mathbf{w}=\hat{\mathbf{w}}}$. Thus, the estimate of the variance of \hat{r}_{tjj^*} is obtained by:

$$\widehat{\text{Var}}(\hat{r}_{tjj^*}) = \mathbf{J}_{tjj^*}^T \hat{\Sigma} \mathbf{J}_{tjj^*}$$

We obtain a 95% confidence interval for r_{tjj^*} as $\left(\hat{r}_{tjj^*} - 1.96\sqrt{\widehat{\text{Var}}(\hat{r}_{tjj^*})}, \hat{r}_{tjj^*} + 1.96\sqrt{\widehat{\text{Var}}(\hat{r}_{tjj^*})}\right)$. We exponentiate \hat{r}_{tjj^*} and its corresponding 95% confidence interval to obtain the estimate and the 95% confidence interval for the hazard ratio, R_{tjj^*} . We illustrate the use of the **straweib** R package for obtaining hazard ratios and corresponding confidence intervals in Section 4.

Comparison of models implemented in packages survival and straweib

In this section, we compare the stratified Weibull regression model implemented in the **survival** package to that implemented in our package, **straweib**.

In the absence of stratification, both models are identical and reduce to the Weibull PH model. However, in the presence of a stratification factor, the models implemented by **survival** and **straweib** correspond to different models, resulting in different likelihood functions and inference. As we discussed in Section 2, the hazard ratio between two subjects with different covariate values within same stratum depends on their stratum in the model implemented in the R package **survival** (Equation (5)), whereas the hazard ratio comparing two individuals within the same stratum is invariant to stratum in the model implemented in the R package **straweib** (Equation (7)). In particular, the Weibull model implemented in the **straweib** shares similarities with the semi-parametric, stratified Cox model for right censored data.

To illustrate the difference between the models implemented in the R packages **survival** and **straweib**, we conducted a simulation study in which 1000 datasets were simulated under the model assumed in the **straweib** package (Equation (6)). For each simulated dataset, since both models have the same number of unknown parameters, we compare the values of the log-likelihood evaluated at the MLEs. Datasets were simulated based on the assumptions that there are 3 strata, each with a 100 subjects; the shape parameters (γ) in the three strata were set to 1.5, 2, and 1, respectively; the baseline scale parameters in the three strata (λ) were set to 0.01, 0.015, and 0.02, respectively. We assumed that there are two independent explanatory variables available for each subject, randomly drawn from $N(0, 1)$ random variables. The coefficients corresponding to each of the two covariates were set to 0.5 and 1, respectively. To simulate interval censored outcomes, we first simulated the true event time for each subject by sampling from a Weibull distribution with the appropriate parameters. We assumed that each subject has 20 equally spaced diagnostic tests, at which the true event status is observed. Each test has a probability of 70% being missing. To obtain the maximum likelihood estimates under each model, we used the `survreg` function in the R package **survival** and the `icweib` function in the **straweib** package.

Figure 1 compares the maximized value of the log-likelihoods under both models, when the data are generated using a simulation mechanism that corresponds to the model implemented in the R package **straweib**. The maximized value of the log-likelihood from the R package **survival** is lower than that from the R package **straweib** for 93.1% of simulated datasets. This is expected as in this simulation study the data generating mechanism is identical to the model implemented in the R package **straweib**. In applications where the proportional hazards assumption is questionable, we recommend fitting both models and comparing the resulting maximized values of the log likelihood.

Whether one model is better than another depends on the data.

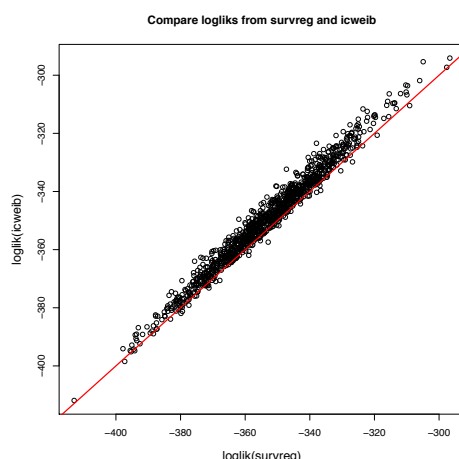


Figure 1: Comparing the maximized values of the log-likelihood obtained from the models implemented in the R package **survival** (X axis) to that from the R package **straweib** (Y axis), when the data is simulated under the model implemented in the R package **straweib**

Example

We illustrate the R package **straweib** with data from a study on the timing of emergence of permanent teeth in Flemish children in Belgium (Leroy et al., 2003). The data analyzed were from the Signal-Tandmobiel project (Vanobbergen et al., 2000), a longitudinal oral health study in a sample of 4430 children conducted between 1996 and 2001. Dental examinations were conducted annually for a period of 6 years and tooth emergence was recorded based on visual inspection. As in Gomez et al. (2009), we will illustrate our R package by analyzing the timing of emergence of the permanent upper left first premolars. As dental exams were conducted annually, for each child, the timing of tooth emergence is known up to the interval from the last negative to the first positive dental examination.

```
data(tooth24)
head(tooth24)
```

	id	left	right	sex	dmf
1	1	2.7	3.5	1	1
2	2	2.4	3.4	0	1
3	3	4.5	5.5	1	0
4	4	5.9	Inf	1	0
5	5	4.1	5.0	1	1
6	6	3.7	4.5	0	1

The dataset is formatted to include 1 row per child. The variable denoted **id** corresponds to the ID of the child, **left** and **right** correspond to the left and right endpoints of the censoring interval in years, **sex** denotes the gender of the child (0 = boy, and 1 = girl), and **dmf** denotes the status of primary predecessor of the tooth (0 = sound, and 1 = decayed or missing due to caries or filled). Right censored observations are denoted by setting the variable **right** to "Inf".

In our analysis below, we use the function `icweib` in the package **straweib**, to fit a stratified Weibull regression model, where the variable **dmf** is the stratum indicator (*S*) and the variable **sex** is an explanatory variable (*Z*).

```
fit <- icweib(L = left, R = right, data = tooth24, strata = dmf, covariates = ~sex)
fit
```

Total observations used: 4386. Model Convergence: TRUE

Coefficients:

```

      coefficient      SE      z p.value
sex      0.331 0.0387 8.55      0

```

```
Weibull parameters - gamma(shape), lambda(scale):
```

```

straname strata gamma  lambda
dmf      0    5.99 1.63e-05
dmf      1    4.85 1.76e-04

```

```
Test of proportional hazards for strata (H0: all strata's shape parameters are equal):
```

```

      test TestStat df  p.value
Wald      44.2   1 2.96e-11
Likelihood Ratio      44.2   1 3.00e-11

```

```

Loglik(model)= -5501.781  Loglik(reduced)= -5523.87
Loglik(null)= -5538.309  Chisq= 73.05611  df= 1  p.value= 0

```

The likelihood ratio test of the PH assumption results in a p value of $3.00e-11$, indicating that the PH model is not appropriate for this dataset. Or in other words, the data suggest that the hazard functions corresponding to the strata defined by $dmf = 0$ and $dmf = 1$ are not proportional. From the stratified Weibull regression model, the estimated regression coefficient for **sex** is 0.331, corresponding to a hazard ratio of 1.39 (95% CI: 1.29 - 1.50). In the output above, the maximized value of the log likelihood of the null model corresponds to the model stratified by covariate **dmf** but excluding the explanatory variable **sex**.

The p value from the Wald test of the null hypothesis of no effect of gender results in a p value of approximately 0 ($p < 10^{-16}$), which indicates that the timing of emergence of teeth is significantly different between girls and boys.

To test the global null hypothesis that both covariates **sex** and **dmf** are not associated with the outcome (time to teeth emergence), we obtain the log-likelihood for global null model, as shown below.

```

fit0 <- icweib(L = left, R = right, data = tooth24)
fit0

```

```
Total observations used: 4386. Model Convergence: TRUE
```

```
Weibull parameters - gamma(shape), lambda(scale):
```

```

straname strata gamma  lambda
strata   ALL    5.3 7.78e-05

```

```

Loglik(model)= -5596.986
Loglik(null)= -5596.986

```

The likelihood ratio test testing the global null hypothesis results in a test statistic $T = -2(l_R - l_F) = -2(-5596.986 + 5501.781) = 190.41$, which follows a χ^2_3 distribution under H_0 , resulting in a p value of approximately 0 ($p < 10^{-16}$).

We illustrate the `HRatio` function in the **straweib** package to estimate the hazard ratio and corresponding 95% confidence intervals for comparing boys without tooth decay ($dmf = 0$) to boys with evidence of tooth decay ($dmf = 1$), where the hazard ratio is evaluated at various time points from 1 through 7 years.

```
HRatio(fit, times = 1:7, NumStra = 0, NumZ = 0, DemStra = 1, DemZ = 0)
```

```

time NumStra DemStra beta*(Z1-Z2)      HR      low95      high95
1    1      0      1      0.1143698 0.06596383 0.1982972
2    2      0      1      0.2520248 0.18308361 0.3469262
3    3      0      1      0.4000946 0.33112219 0.4834339
4    4      0      1      0.5553610 0.49863912 0.6185351
5    5      0      1      0.7162080 0.66319999 0.7734529
6    6      0      1      0.8816470 0.79879884 0.9730878
7    7      0      1      1.0510048 0.91593721 1.2059899

```

The output indicates that the hazard ratio for boys comparing the stratum $dmf = 0$ to stratum $dmf = 1$ is small initially (e.g. 0.11 at 1 year) but tends to 1 in later years (e.g. 0.88 at 6 years and 1.05 at 7

years). Prior to 6 years, the hazard ratio is significantly less than 1, indicating that the timing of teeth emergence is delayed in children with tooth decay ($dmf = 1$) when compared to children without tooth decay ($dmf = 0$).

We illustrate estimation of the survival function in Figure 2 by plotting the survival functions and corresponding 95% point wise confidence intervals for girls ($Z = 1$), with and without tooth decay .

```
plot(fit, Z = 1, tRange = c(1, 7), xlab = "Time (years)", ylab = "Survival Function",
     main = "Estimated survival function for girls")
```

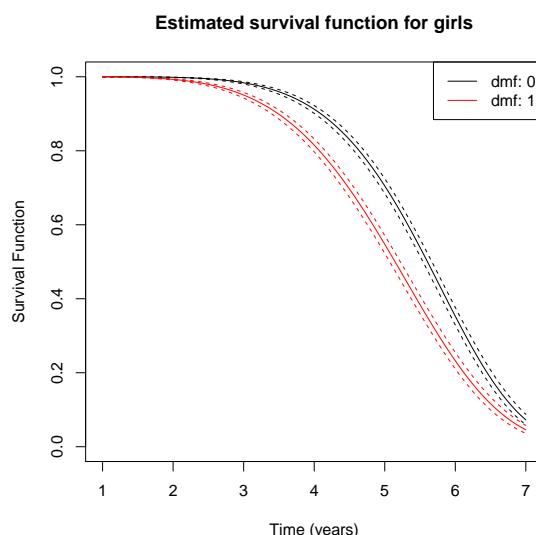


Figure 2: Estimated survival functions for girls, comparing the subgroup with sound primary predecessor of the tooth ($dmf = 0$) to the subgroup with unsound primary predecessor of the tooth ($dmf = 1$).

We compare our results from the **straweib** package to that obtained from the **survival** package.

```
library(survival)
tooth24.survreg <- tooth24
tooth24.survreg$right <- with(tooth24, ifelse(is.finite(right), right, NA))
fit1 <- survreg(Surv(left, right, type="interval2") ~ sex + strata(dmf) + factor(dmf),
               data = tooth24.survreg)
fit1
```

Call:

```
survreg(formula = Surv(left, right, type = "interval2") ~ sex +
        strata(dmf) + factor(dmf), data = tooth24.survreg)
```

Coefficients:

```
(Intercept)      sex factor(dmf)1
 1.84389938  -0.06254599  -0.06491729
```

Scale:

```
dmf=Sound1 dmf=Sound2
 0.1659477  0.2072465
```

```
Loglik(model)= -5499.3  Loglik(intercept only)= -5576.2
```

```
Chisq= 153.8 on 2 degrees of freedom, p= 0
```

```
n= 4386
```

The maximized value of the log-likelihood from the R package **survival** is -5499.3 (shown below), as compared to the maximized value of the log-likelihood of -5501.8 from the R package **straweib**.

To clarify the specific assumptions made by the models implemented in the **survival** and **straweib** packages, we carried out subgroup analyses in which we fit a Weibull PH model separately to each of the strata $dmf = 0$ and $dmf = 1$. The results from the Weibull PH model fit to the subgroup of children in the $dmf = 0$ stratum is shown below:

```
fit20 <- icweib(L= left, R=right, data=tooth24[tooth24$dmf==0, ], covariates = ~sex)
fit20 ### Partial results shown below
Coefficients:
      coefficient      SE      z  p.value
sex           0.448 0.0543 8.25 2.22e-16
```

The results from the Weibull PH model fit to the subgroup $dmf = 1$ is shown below:

```
fit21 <- icweib(L= left, R=right, data=tooth24[tooth24$dmf==1, ], covariates = ~sex)
fit21 ### Partial results shown below
Coefficients:
      coefficient      SE      z  p.value
sex           0.208 0.0554 3.76 0.000169
```

The model using the PH scale (implemented by **straweib** package) replaces the stratum specific hazard ratios for sex of $e^{0.448} = 1.57$ for the subgroup $dmf = 0$ and $e^{0.208} = 1.23$ for the subgroup $dmf = 1$ with a common value, $e^{0.331} = 1.39$.

Since the Weibull distribution has both the PH and accelerated failure time (AFT) property (Collett, 2003), the identical set of subgroup analyses can be fit using the **survival** package. Results from the fit using the **survival** package for the subgroup $dmf = 0$ are shown below:

```
fit20.survreg <- survreg(Surv(left, right, type="interval2") ~ sex,
                        data = tooth24.survreg[tooth24.survreg$dmf==0, ])
fit20.survreg ### Partial results shown below
Coefficients:
(Intercept)      sex
1.85029150 -0.07453785
```

Similar results using the **survival** package for the subgroup $dmf = 1$ are shown below:

```
fit21.survreg <- survreg(Surv(left, right, type="interval2") ~ sex,
                        data = tooth24.survreg[tooth24.survreg$dmf==1, ])
fit21.survreg ### Partial results shown below
Coefficients:
(Intercept)      sex
1.76931556 -0.04303767
```

In particular, the model assuming a common sex coefficient in the AFT scale (implemented by **survival** package) replaces the value of sex coefficient -0.075 for the subgroup with $dmf = 0$ and sex coefficient of -0.043 for the subgroup $dmf = 1$ with a shared common value, -0.063 .

To assess the goodness of fit of the stratified Weibull model implemented by **straweib**, we created a multiple probability plot, as described in chapter 19 of Meeker and Escobar (1998). This diagnostic plot was created by splitting the dataset into 4 subgroups based on the values of **sex** and **dmf**. Within each group, we estimated the cumulative incidence at each visit time using a non-parametric procedure for interval censored data (Turnbull, 1976). The non-parametric estimates of cumulative incidence within each subgroup were compared to that obtained from the stratified Weibull model implemented by **straweib** package. We use the R package **interval** (Fay and Shaw, 2010) to obtain Turnbull's NPMLE estimates and the R package **straweib** for the estimates from the stratified Weibull model (code available upon request). Figure 3 shows the diagnostic plot.

Table 1 presents the estimates of hazard ratio for **sex**, within each of the strata defined by $dmf = 0$ and $dmf = 1$, comparing three different analyses - (1) Using the **survival** package to stratify on the variable **dmf** and including **sex** as an explanatory variable; (2) Using the **straweib** package to stratify on the variable **dmf** and including **sex** as an explanatory variable; (3) Fitting a Weibull PH model with **sex** as an explanatory variable, separately within each of the two subgroups defined by $dmf = 0$ and $dmf = 1$.

```
HR.straweib <- exp(fit$coef[1, 1])
HR.survreg <- exp(-fit1$coefficients['sex']/fit1$scale)
HR.subgroup <- exp(c(fit20$coef[1, 1], fit21$coef[1, 1]))
```

Concluding remarks

We have developed and illustrated an R package **straweib** for the analysis of interval-censored outcomes, based on a stratified Weibull regression model. The proposed model shares similarities with the semi-parametric stratified Cox model. We illustrated the R package **straweib** using data from

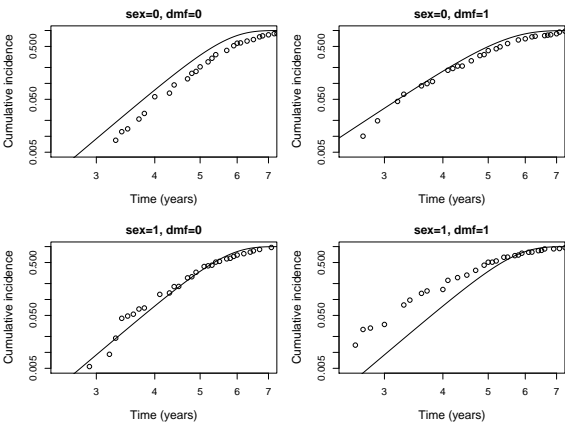


Figure 3: Comparing non-parametric (points) and Weibull model (lines) based estimates of cumulative incidence within each group based on covariates **sex** and **dmf**.

Table 1: Hazard ratio estimates for gender, comparing the models implemented in the R packages **survival**, **straweib** and subgroup analyses

stratum	Package survival	Package straweib	Stratum specific subgroup analyses
dmf = 0	1.46	1.39	1.56
dmf = 1	1.35	1.39	1.23

a prospective study on the timing of emergence of permanent teeth in Flemish children in Belgium (Leroy et al., 2003).

Although the models and R package are illustrated for the analysis of interval-censored time-to-event outcomes, the methods proposed here are equally applicable for the analysis of right-censored outcomes. The syntax for the analysis of right-censored observations is explained in the manual accompanying the **straweib** package available on CRAN (Gu and Balasubramanian, 2013).

Acknowledgements

This research was supported by NICHD grant R21 HD072792.

Bibliography

D. Collett. *Modelling Survival Data in Medical Research, Second Edition*. Texts in statistical science. Taylor & Francis, 2003. ISBN 9781584883258. [p32, 33, 38]

D. R. Cox and D. V. Hinkley. *Theoretical Statistics*. Chapman and Hall, 1979. [p33]

D. T. Dunn, R. J. Simonds, M. Bulterys, L. A. Kalish, J. Moye, A. de Maria, C. Kind, C. Rudin, E. Denamur, A. Krivine, C. Loveday, and M. L. Newell. Interventions to prevent vertical transmission of HIV-1: effect on viral detection rate in early infant samples. *AIDS*, 14(10):1421–1428, 2000. [p31]

M. P. Fay and P. A. Shaw. Exact and asymptotic weighted logrank tests for interval censored data: The interval R package. *Journal of Statistical Software*, 36(2):1–34, 2010. URL <http://www.jstatsoft.org/v36/i02/>. [p38]

D. M. Finkelstein. A proportional hazards model for interval-censored failure time data. *Biometrics*, 42(4):845–854, 1986. [p31]

E. Goetghebeur and L. Ryan. Semiparametric regression analysis of interval-censored data. *Biometrics*, 56(4):1139–1144, 2000. [p31]

G. Gomez, M. L. Calle, R. Oller, and K. Langohr. Tutorial on methods for interval-censored data and their implementation in R. *Statistical Modelling*, 9(4):259–297, 2009. [p31, 35]

- X. Gu and R. Balasubramanian. *straweib: Stratified Weibull Regression Model*, 2013. URL <http://CRAN.R-project.org/package=straweib>. R package version 1.0. [p31, 33, 39]
- R. Leroy, K. Bogaerts, E. Lesaffre, and D. Declerck. The emergence of permanent teeth in flemish children. *Community Dentistry and Oral Epidemiology*, 31(1):30–39, 2003. [p31, 35, 39]
- J. C. Lindsey and L. M. Ryan. Tutorial in biostatistics - methods for interval-censored data. *Statistics in Medicine*, 17(2):219–238, 1998. [p31]
- W. Meeker and L. Escobar. *Statistical Methods for Reliability Data*. Wiley Series in Probability and Statistics. Wiley, 1998. ISBN 9780471673279. [p38]
- T. Therneau. *A Package for Survival Analysis in S*, 2012. R package version 2.36-14. [p31, 32]
- B. W. Turnbull. Empirical distribution function with arbitrarily grouped, censored and truncated data. *Journal of the Royal Statistical Society Series B-Methodological*, 38(3):290–295, 1976. [p31, 38]
- J. Vanobbergen, L. Martens, E. Lesaffre, and D. Declerck. The Signal-Tandmobiel project a longitudinal intervention health promotion study in flanders (belgium): baseline and first year results. *Eur J Paediatr Dent*, 2:87–96, 2000. [p35]

Xiangdong Gu
Division of Biostatistics and Epidemiology
University of Massachusetts, Amherst, MA, USA
xdgu@schoolph.umass.edu

David Shapiro
Department of Biostatistics, Harvard School of Public Health, Boston, MA, USA
shapiro@sdac.harvard.edu

Michael D. Hughes
Department of Biostatistics, Harvard School of Public Health, Boston, MA, USA
mhughes@hsph.harvard.edu

Raji Balasubramanian
Division of Biostatistics and Epidemiology
University of Massachusetts, Amherst, MA, USA
rbalasub@schoolph.umass.edu

brainR: Interactive 3 and 4D Images of High Resolution Neuroimage Data

by John Muschelli, Elizabeth Sweeney, and Ciprian Crainiceanu

Abstract We provide software tools for displaying and publishing interactive 3-dimensional (3D) and 4-dimensional (4D) figures to html webpages, with examples of high-resolution brain imaging. Our framework is based in the R statistical software using the **rgl** package, a 3D graphics library. We build on this package to allow manipulation of figures including rotation and translation, zooming, coloring of brain substructures, adjusting transparency levels, and addition/or removal of brain structures. The need for better visualization tools of ultra high dimensional data is ever present; we are providing a clean, simple, web-based option. We also provide a package (**brainR**) for users to readily implement these tools.

Introduction

We provide a set of software tools that produce interactive 3-dimensional (3D) and 4-dimensional (4D) figures using currently available open-source software. These images can be embedded into webpages (<http://bit.ly/1kEJH6q>), creating an environment where users can interact with the images without additional software or expertise. We present two examples of applications of these tools using neuroimaging data: 1) a 3D example: displaying hyper-intense white matter regions in a template image; and 2) a 4D example: tracking the lesion formation process over time via 3D segmentation.

Many applications display true 3D data as single 2-dimensional (2D) cross-sections, a series of cross-sections, or a projection of the data onto 2D space. In neuroimaging data, for example, the current standard is to present spatial results as a set of 2D figures, or slices. Figure 1 displays the magnetic resonance image (MRI) of a normal brain and includes three planar views: axial (vertical plane dividing the body into front and back), sagittal (vertical plane dividing the brain into right and left), and coronal (horizontal plane dividing the brain into top and bottom). Other types of presentations can be employed, such as lightbox views where all slices of the brain are presented in series, single plane views, 3D surface images, or maximum intensity projections onto transparent cross-sectional views (known as “glass brains”, Figure 1B). We wish to improve on these data representations by using truly 3D figures.

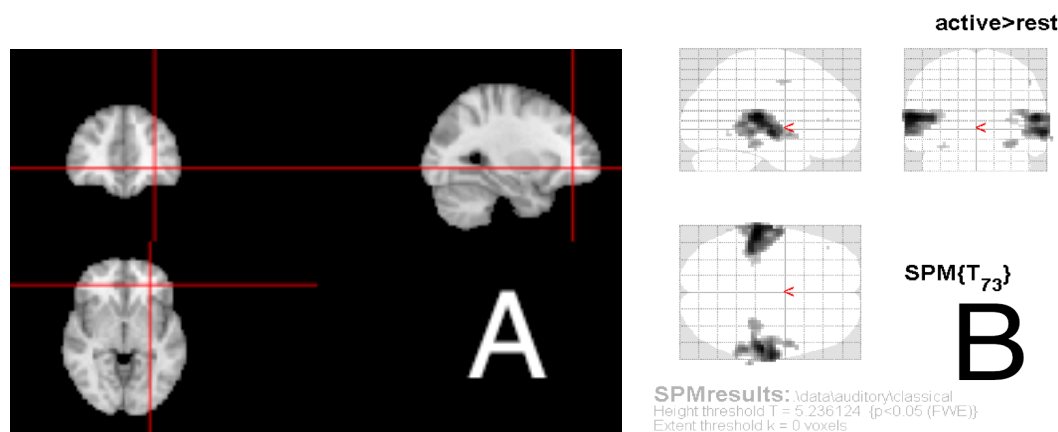


Figure 1: Example of an orthographic representation of the brain (A), where the brain is displayed axially (bottom), coronally (top left), and sagittally (top right). Cross-hairs (red) are shown to orient the user to how certain areas fall in each dimension. Figures using maximum intensity projections show a similar set of 3 transparent views, known as “glass brains”, such as (B) from the SPM8 manual, showing how certain information (in this case activation) corresponds spatially (Ashburner et al., 2008).

The goal of these multi-panel 2D figures is to allow the user to: 1) orient themselves spatially; 2) understand the local brain features or pathology in relationship to known and observable biological landmarks; and 3) provide qualitative or quantitative assessments of the image. While these approaches provide important information and are extensively used in practice, they also have innate limitations. For example, the slices presented must be chosen by the presenter/author, which may not

be representative of all areas of the brain. Most importantly though, the true 3D structure of the data is not exploited and 4D interactive visualizations are difficult to implement.

With the advancement of 3D rendering techniques and web-based tools, the ability to provide more intuitive and informative figures is now possible. People are not yet taking full advantage of the ability to present true, interactive 3D figures. This may be because 3D figures are not currently accepted for publication, a fact that is likely to change with the generational shift to digital journal print. Therefore, the purpose of this article is to inform the reader how to create and use interactive 3D figures for exploration, presentation, and published-article figures.

We will be providing 1) the characteristics of a good 4D interactive figure, 2) a framework for creating these figures in R, 3) examples of creating figures within this framework using R, and 4) description of practical applications for these figures. The examples in the article are brain images, but these concepts and methods can be used for general 3D surfaces.

What makes a good 4D interactive figure?

In order to determine software necessary for 3D figure generation, we discuss properties of the figures and the process that are required for their generation. We argue that for the interactive capabilities of 3D figures to have added value over current presentation methods they should have the following characteristics:

- Rotation/Translation: allows the user to see objects from multiple angles and positions.
- Digital zoom: enables multilayer display of information from minute details to coarser anatomy.
- Color: allows for proper contrast between structures, and also the ability to highlight regions of interest.
- Transparency: provides the ability to adaptively explore the changing opacity of cortical surface as well as sub-cortical deep structures.
- Addition/Removal of structures: allows users to contrast one surface with another by adding or removing an individual surface to see how the surfaces overlap and compare spatially.

These capabilities provide users the ability to more accurately view differences between populations, image processing pipelines, and segmentation algorithms. With the addition and removal of surfaces, users can also view changes in brain structures longitudinally in 4D.

We refer to 4D viewing as observing 3D objects over another domain other than space, such as time. The user interacts over this domain while keeping the brain in the same 3D space. For example, characterizing the size and shape changes of the hippocampus over time within a person. Similarly, activation maps derived from a functional MRI (fMRI) task can be calculated for a diseased group of people and a control group. We can present the area of activation from the control group and switch to the map for a diseased group for comparison. Lastly, we have used these techniques to determine how a hemorrhage in the brain changes over time—pre and post treatment—depicting where the treatment had removed hemorrhage.

Even with these capabilities, without quick and easy usability, sophisticated plotting tools, such as those described here, will not be used in practice. Therefore, the publishing platform needs to render surfaces very fast and be easy to use and manipulate. Publishing and presenting such figures in a user-friendly interface that does not require third-party software is another essential part of visualization.

Software and tools

Many rendering systems, or analysis tools with rendering capabilities exist, such as FreeSurfer (<http://surfer.nmr.mgh.harvard.edu/>), ParaView (Henderson et al., 2004), MIPAV (McAuliffe et al., 2013), 3D Slicer (Pieper et al., 2004), and **rgl** (Adler and Murdoch, 2014) in R. We will focus on the R package **rgl** as it has the desired characteristics outlined above and can easily export images to the web with a small amount of setup and knowledge of the system.

The R programming language is becoming more widely used, with expanding areas of research and capabilities. **rgl** is an R implementation of the graphics programming interface Open Graphics Library (OpenGL; Shreiner et al., 2009), that allows for 3D rendering. These rendered objects can be interactively explored on screen. We prefer using R because it has many packages that read, analyze, and display neuroimaging data (Whitcher et al. 2011; Tabelow and Polzehl 2011; Bordier et al. 2011, see also the CRAN Task View on Medical Image Analysis; Whitcher 2013), is multi-platform, free, and open-source with a large user and developer community. As a statistical programming language, it contains state-of-the-art statistical tools with a growing number of packages for additional analysis,

and the capability to create highly customizable figures. Our framework uses **rgl** to generate the figures and export these figures as webpages using WebGL, the web implementation of OpenGL. Using these objects and the X Toolkit (XTK; <https://github.com/xtk/X>), one can create customizable webpages with versatile user-interaction capabilities. Moreover, R also has integrated tools such as **knitr** and **Sweave** that allow the user to completely reproduce a figure and analysis, and **slidify** to create HTML5 slide decks with these figures embedded (Xie, 2014; Leisch, 2002; Vaidyanathan, 2012).

Levine et al. (2010) presented a way to embed 3D objects from **rgl** directly into a Portable Document Format (PDF) document, along with additional applications (Bowman and Shardt, 2009). Though this option is promising, we have found that large images, such as Figure 3 do not render quickly and can lag when interacting with them on a laptop and sometimes crash before rendering even with moderate amounts of memory (Mac OSX 10.9.2, 16Gb RAM, 2.8GHz Intel i7). Furthermore, all capabilities mentioned above were not implemented, such as interactively changing transparency or 4D visualization, and customized control is not easily implemented in the framework of Levine et al. (2010) without learning additional graphics languages. Some of these are current limitations of the interactive capabilities within PDF, but the integration of JavaScript into PDF documents will likely expand on these capabilities (Story, 2001). We acknowledge others have proposed similar ideas of 3D rendering in R using Virtual Reality Markup Language (VRML; Glaab et al., 2010). We chose to focus on WebGL because browsers require VRML viewer browser plugins and popular browsers (Safari, Google Chrome, Firefox) can render WebGL without any additional plugins.

Although users may prefer some of the aforementioned systems, we wish to inform the community about a simple option to create these figures; we also discuss why they should be more generally used and accepted for publication.

Framework

Figure 2 illustrates our working framework for figure generation. Briefly, the neuroimaging data is processed and areas of interest are extracted. Examples include: activation maps from fMRI studies during tasks, parcellations of cortical and deep structures, segmentation maps for detection of abnormal tissue, and more general regions of interest (ROIs). These images are passed into the rendering system and surfaces are created from either binary masks or thresholds of the image values. A surface is a collection of vertices that are connected. A collection of multiple surfaces together will be referred to as a scene.

First, the **misc3d** package creates 3D contours for the surfaces for rendering (Feng and Tierney, 2008). Once the surface is created, **rgl** can create and export the scene in two ways: 1) output all properties of the scene (colors, shading and lighting, surface orientation) and vertices of the surface directly into one html file or 2) export the vertices of a surface into a 3D-object file and separately write the scene properties into the html file. Option 1 is easier and is our recommendation for exporting 3D images. This process uses the `writeWebGL` command from the **rgl** package and creates one html file with an accompanying JavaScript library. An example of an exported figure is shown in Figure 3, which depicts a 3D rendering of a brain template (Grabner et al., 2006) and areas of hyper-intense white matter in WebGL.

One problem with this option is that surfaces in WebGL are limited to 65535 vertices, which may be too few for some large figures that are common in high resolution imaging data. To avoid this, users can downsample their data to reduce the dimensions or use `writeWebGL_split`: a simple function to split the data into smaller structures so that these larger images can be displayed (courtesy of Duncan Murdoch). This function is located in the **brainR** package. Although easily executed, this option is limited in 2 ways: 1) the html file contains all vertices of a surface; there can be thousands of vertices—this limits the readability of the html file and 2) it does not easily allow for adding user control of surfaces to create a 4D figure.

```
require(brainR)

# Template from MNI152 from McGill
template <- readNIfTI(system.file("MNI152_T1_2mm_brain.nii.gz", package = "brainR"),
  reorient = FALSE)

### 4500 - value that empirically value that presented a brain with gyri
### lower values result in a smoother surface
dtemp <- dim(template)
contour3d(template, level = 4500, alpha = 0.1, draw = TRUE)
```

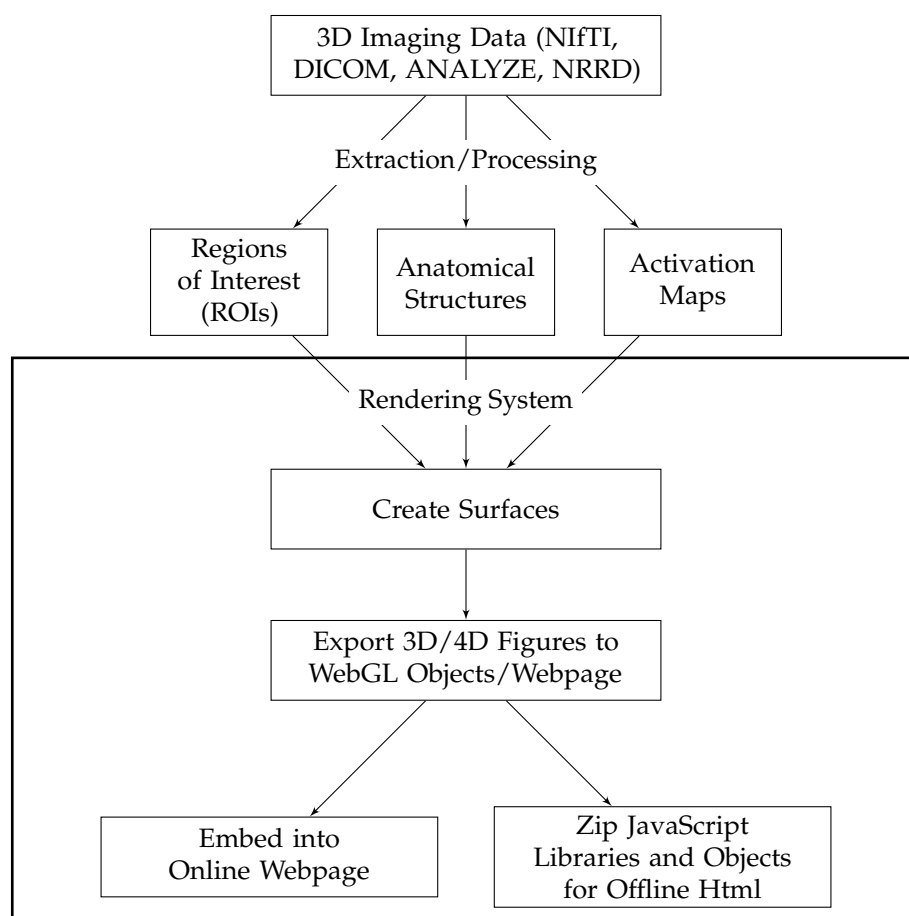


Figure 2: Overall workflow of generating 3D or 4D figures from data. After the raw data has been processed and areas have been extracted, surfaces can be rendered. We are concerned with the steps outlined in the large box: creating surfaces and export to the web. The last branch shows 2 options for export: publishing the figure to the web or enclosing it in a folder with all other libraries needed for rendering. The second option allows users to include these zipped directories as supplementary figures (3D and 4D interactive images are not currently published by journals).

```

### this would be the 'activation' or surface you want to render
### - hyper-intense white matter
contour3d(template, level = c(8200, 8250), alpha = c(0.5, 0.8),
          add = TRUE, color = c("yellow", "red"))
# create text for orientation of right/left
text3d(x = dtemp[1]/2, y = dtemp[2]/2, z = dtemp[3] * 0.98, text = "Top")
text3d(x = dtemp[1] * 0.98, y = dtemp[2]/2, z = dtemp[3]/2, text = "Right")

### render this on a webpage and view it!
browseURL(paste("file://",
               writeWebGL_split(dir = file.path(tempdir(), "webGL"),
                                template = system.file("my_template.html", package = "brainR"),
                                width = 500), sep = ""))
  
```

Option 2 allows users to export individual objects/surfaces to either STL (STereoLithography) or Wavefront OBJ files—file formats for 3D surfaces. XTK renders these individual surfaces, and users interact with these objects by turning surfaces on or off, changing opacity, or manipulating any of the surface attributes interactively on the webpage with JavaScript. This process produces interactive 4D figures, allowing 3D brain structures to be contrasted such as parcellations of the same brain area under different algorithms or single-subject stroke segmentations over time.

We have included functions (`write4D` and `write4D.file`) in the **brainR** package that allow users to export a scene and embed it in html with basic JavaScript controls. We have included a 4D example of segmentation of white matter lesions in a patient with multiple sclerosis (MS) using SubLIME, an automated method for segmenting enlarging and incident MS lesions (Sweeney et al., 2012). Each

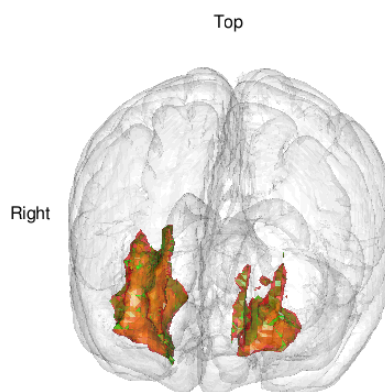


Figure 3: Snapshot of a 3D interactive figure exported from R using `writeWebGL`. The figure has many of the qualities of a good figure: transparency, text rendering, colors, and multiple surfaces. The online web version is located <http://bit.ly/QrJqs2> (high-resolution version). Viewing the page source code illustrates how all points of the surfaces are outputted directly to html—rendering users unable to easily replicate the figure without source code. Full code is located at <http://bit.ly/1gE48M1>.

surface represents enlarging lesions compared to the last visit. The rendered html allows control of the opacity of the brain and to compare which areas of the brain have lesion enlargements over time (Figure 4). This 4D rendering of new and enlarging MS lesions could be used to monitor the clinical progression of the disease.

```
### Example data courtesy of Daniel Reich
### Each visit is a binary mask of lesions in the brain
template <- readNIfTI(system.file("MNI152_T1_1mm_brain.nii.gz",
                                package = "brainR"), reorient = FALSE)
brain <- contour3d(template, level = 4500, alpha = 0.8, draw = FALSE)
imgs <- paste("Visit_", 1:5, ".nii.gz", sep = "")
files <- sapply(imgs, system.file, package = "brainR")
scene <- list(brain)
## loop through images and threshold masks
nimgs <- length(imgs) # get number of images
cols <- rainbow(nimgs) # set colors
for (iimg in 1:nimgs) {
  mask <- readNIfTI(files[iimg], reorient = FALSE)[,,1] # read image mask
  ## use 0.99 for level of mask - binary
  activation <- contour3d(mask, level = 0.99, alpha = 1, add = TRUE,
                        color = cols[iimg], draw = FALSE)
  ## add these triangles to the list
  scene <- c(scene, list(activation))
}
## make output image names from image names
fnames <- c("brain.stl", gsub(".nii.gz", ".stl", imgs, fixed = TRUE))
outfile <- "index_4D_stl.html"

## write the html file out with JavaScript checkboxes
write4D(scene = scene, fnames = fnames, outfile = outfile, standalone = TRUE)
browseURL(outfile)
```

Discussion

Webpages as a medium

Our main proposal is to create interactive 3D and 4D figures that allow easy manipulation of embedded 3D objects and additional annotations. We propose export directly to webpages, which makes the figures widely available, reduces dependence on operating systems, and allows a much larger community to be able to interact with the data.

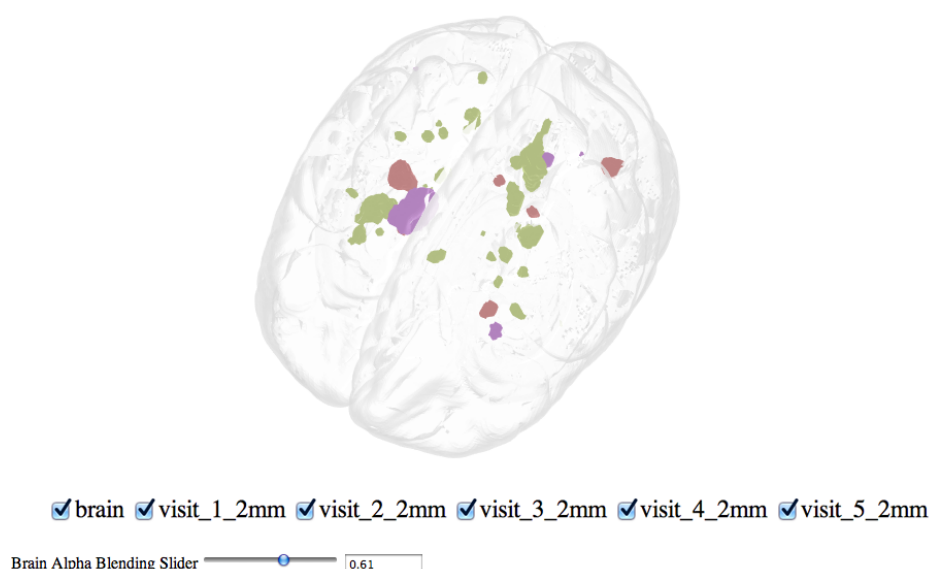


Figure 4: Example of a 4D Figure: a 3D rendered brain (gray) and areas of enlarging white matter lesions in a patient with MS over different followup visits (multiple colors). The checkboxes switch between lesion enlargements at different followup visits and can display multiple visits simultaneously. The brain alpha blending slider (bottom left) allows you to change the opacity of the brain image; this feature can also be added for each surface. We observe that lesions enlarge at many different regions over different visits. The interactive version is located at <http://bit.ly/1kEJH6q>, and code <http://bit.ly/1kdnWeg>.

Browser interfaces are versatile, flexible, and have large support communities, but rendering may rely on libraries that are not maintained by researchers. Thus, their maintenance is not guaranteed, but the likelihood of maintenance is higher with increased use. The main concern is to produce figures under JavaScript libraries and R packages that are popular and therefore more likely to be updated and improved. Also, including these libraries in the document as supplementary material for standalone figures allows for authors to guarantee which versions of the software readers will be using.

Although highly versatile, web-based tools are not browser independent. Some 3D and 4D interactive figures may be supported on some browsers but not others and may not be usable on the expanding number of devices, such as tablets or smart phones. Our current software works with browsers supporting WebGL and HTML5. Even with these limitations, webpages as figures are extremely powerful and adaptable and present the basic avenue for easy-to-use interactive figures.

Embedding figures into PDF documents

Previous authors have contributed work that allow users to embed 3D figures directly into PDF documents (Levine et al., 2010; Barnes et al., 2013). While we think this is a useful endeavor for many applications, some of the capabilities described above are lacking. Another drawback is that predominantly all rendering must be done within the Adobe Acrobat Reader (Adobe Systems, 2014), as it is the primary PDF reader with 3D capabilities. Also, we have found that web-based rendering performs much faster than those within PDF for the dimensionality of points we are plotting. Overall, we think this system may fully integrate the needs of the end user into one document, but currently is not optimal for 3D and 4D neuroimaging figures.

Practical applications of 3/4D figures

Above, we described how this framework has been used to view a longitudinal image segmentation in a holistic way for scans of patients with MS. Other potential applications for this framework are exploratory data analysis and communication of analysis results. The results from analysis such as segmentation, thresholding, or any other surface-generating mechanism can be easily viewed in 3D. Parameters from an analysis can be interactively changed and optimized in this framework. Also, as these figures are in webpages, embedding them into online presentations or journal articles is straightforward and simple. Therefore, journals will not be required to change the way they publish articles while still allowing audiences to interact with figures and get a more comprehensive view of results.

Conclusion

Interactive 3D figures can be created quickly. They are reproducible and can be exported easily to standalone webpages. This framework allows for these figures to supplement 2D figures to present results from analysis and image processing in neuroimaging research. These figures allow users to view the data in the natural setting: 3D space. Users can also create 4D images that can spatially display longitudinal data, group comparisons of areas of activation, differences between parcellation schemes, different independent component analysis networks, and other results from neuroimaging analysis. This provides a powerful alternative to the current state-of-the-art, which relies predominantly on projections, multiple figures, or other reductions of data that make comparisons more difficult than necessary. To our knowledge we are the first to present a framework for depicting neuroimaging results in a condensed, interactive figure with the aforementioned capabilities.

Sources of funding

The project described was supported by the National Institutes of Health (NIH) grant RO1EB012547 from the National Institute of Biomedical Imaging And Bioengineering, training grant T32AG000247 from the National Institute on Aging, NIH grants RO1NS060910 and RO1NS085211 from the National Institute of Neurological Disorders and Stroke (NINDS), and by NIH grant RO1MH095836 from the National Institute of Mental Health.

Acknowledgements

The authors thank Brian Caffo, Martin Lindquist, and Duncan Murdoch for comments/help and Daniel Reich for providing the MS lesion data.

Supplemental material

Enabling WebGL in Safari

By default, WebGL is not enabled in Safari. This is a description of how to enable WebGL in Safari in order to view the webpages created in this paper.

In Safari, open the Safari menu and select Preferences. Then, click the Advanced tab in the Preferences window. Then, at the bottom of the window, check the Show Develop menu in the menu bar checkbox. Then, open the Develop menu in the menu bar and select Enable WebGL.

Bibliography

- D. Adler and D. Murdoch. *rgl: 3D Visualization Device System (OpenGL)*, 2014. URL <http://CRAN.R-project.org/package=rgl>. R package version 0.93.996. [p42]
- Adobe Systems. Adobe acrobat reader, 2014. URL www.adobe.com/acrobat. [p46]
- J. Ashburner, G. Barnes, C. Chen, J. Daunizeau, G. Flandin, K. Friston, D. Gitelman, S. Kiebel, J. Kilner, V. Litvak, et al. *SPM8 Manual*. Functional Imaging Laboratory, Institute of Neurology, 2008. [p41]
- D. G. Barnes, M. Vidiassov, B. Ruthensteiner, C. J. Fluke, M. R. Quayle, and C. R. McHenry. Embedding and publishing interactive, 3-dimensional, scientific figures in portable document format (PDF) files. *PLoS ONE*, 8(9):e69446, 2013. [p46]
- C. Bordier, M. Dojat, and P. L. de Micheaux. Temporal and spatial independent component analysis for fMRI data sets embedded in the AnalyzeFMRI R package. *Journal of Statistical Software*, 44(9): 1–24, 2011. URL <http://www.jstatsoft.org/v44/i09/>. [p42]
- J. C. Bowman and O. Shardt. Asymptote: Lifting TeX to three dimensions. *TUGboat: The Communications of the TEX Users Group*, 30(1):58–63, 2009. [p43]
- D. Feng and L. Tierney. Computing and displaying isosurfaces in R. *Journal of Statistical Software*, 28(1), 2008. URL <http://www.jstatsoft.org/v28/i01/>. [p43]
- E. Glaab, J. Garibaldi, and N. Krasnogor. *vrmlgen: An R package for 3d data visualization on the web*. *Journal of Statistical Software*, 36(8):1–18, 2010. URL <http://www.jstatsoft.org/v36/i08/>. [p43]

- G. Grabner, A. L. Janke, M. M. Budge, D. Smith, J. Pruessner, and D. L. Collins. Symmetric atlasing and model based segmentation: An application to the hippocampus in older adults. In *Medical Image Computing and Computer-Assisted Intervention—MICCAI 2006*, pages 58–66. Springer, 2006. [p43]
- A. Henderson, J. Ahrens, and C. Law. *The ParaView Guide*. Kitware, Clifton Park, NY, 2004. [p42]
- F. Leisch. Sweave: Dynamic generation of statistical reports using literate data analysis. In *COMPSTAT 2002—Proceedings in Computational Statistics*, pages 575–580, Heidelberg, 2002. Physica-Verlag. [p43]
- R. A. Levine, L. Tierney, H. Wickham, E. Sampson, D. Cook, and D. A. van Dyk. Editorial: Publishing animations, 3d visualizations, and movies in JCGS. *Journal of Computational and Graphical Statistics*, 19(1):1–2, 2010. [p43, 46]
- M. McAuliffe, P. Bazin, A. Bokinsky, R. Cheng, B. Gandler, E. McCreedy, J. Senseney, and B. Tyrie. *MIPAV: Medical Image Processing, Analysis, and Visualization (Version 7.0.1) [Software]*, 2013. Available from <http://mipav.cit.nih.gov/>. [p42]
- S. Pieper, M. Halle, and R. Kikinis. 3D slicer. In *IEEE International Symposium on Biomedical Imaging: Nano to Macro, 2004*, volume 1, pages 632–635, Apr. 2004. [p42]
- D. Shreiner et al. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 3.0 and 3.1*. Addison-Wesley Professional, 2009. [p42]
- D. Story. Techniques of introducing document-level JavaScript into a PDF file from a LaTeX source. *TUGboat: The Communications of the TEX Users Group*, 22(3):161–167, 2001. [p43]
- E. Sweeney, R. Shinohara, C. Shea, D. Reich, and C. Crainiceanu. Automatic lesion incidence estimation and detection in multiple sclerosis using multisequence longitudinal MRI. *American Journal of Neuroradiology*, 34(1):68–73, 2012. [p44]
- K. Tabelow and J. Polzehl. Statistical parametric maps for functional MRI experiments in R: The package fmri. *Journal of Statistical Software*, 44(11):1–21, 2011. URL <http://www.jstatsoft.org/v44/i11/>. [p42]
- R. Vaidyanathan. *slidify: Generate reproducible html5 slides from R markdown*, 2012. URL <http://ramnathv.github.com/slidify/>. R package version 0.3.3. [p43]
- B. Whitcher. CRAN task view: Medical image analysis, 2013. URL <http://CRAN.R-project.org/view=MedicalImaging>. Version 2013-08-12. [p42]
- B. Whitcher, V. J. Schmid, and A. Thornton. Working with the DICOM and NIfTI data standards in R. *Journal of Statistical Software*, 44(6):1–28, 2011. URL <http://www.jstatsoft.org/v44/i06/>. [p42]
- Y. Xie. *knitr: A general-purpose package for dynamic report generation in R*, 2014. URL <http://yihui.name/knitr/>. R package version 1.6. [p43]

John Muschelli
PhD Student
Johns Hopkins Bloomberg School of Public Health,
Baltimore, MD 21231
USA jmuschel@jhsph.edu

Elizabeth Sweeney
PhD Student
Johns Hopkins Bloomberg School of Public Health
Baltimore, MD 21231
USA
Special Volunteer Translational Neuroradiology Unit, Neuroimmunology Branch, National Institute of Neurological Disease and Stroke, National Institute of Health
Bethesda, MD 20892
USA emsweene@jhsph.edu

Ciprian Crainiceanu
Professor
Johns Hopkins Bloomberg School of Public Health,
Baltimore, MD 21231
USA ccrainic@jhsph.edu

The RWiener Package: an R Package Providing Distribution Functions for the Wiener Diffusion Model

by Dominik Wabersich and Joachim Vandekerckhove

Abstract We present the **RWiener** package that provides R functions for the Wiener diffusion model. The core of the package are the four distribution functions `dwiener`, `pwiener`, `qwiener` and `rw Wiener`, which use up-to-date methods, implemented in C, and provide fast and accurate computation of the density, distribution, and quantile function, as well as a random number generator for the Wiener diffusion model. We used the typical Wiener diffusion model with four parameters: boundary separation, non-decision time, initial bias and drift rate parameter. Beyond the distribution functions, we provide extended likelihood-based functions that can be used for parameter estimation and model selection. The package can be obtained via CRAN.

Introduction

The diffusion model is one of the most successful models of choice reaction time in cognitive psychology (see Wagenmakers, 2009, for an overview) and, while software packages that make diffusion model analyses possible have been made available (Vandekerckhove and Tuerlinckx, 2007, 2008; Vandekerckhove et al., 2011; Voss and Voss, 2007; Wagenmakers et al., 2007; Wiecki et al., 2013), a diffusion model R package was so far missing. Due to the nature of the model, no closed analytical forms for the computation of the distribution functions exist, which has generated a need for more involved computational approximations of the various functions (Blurton et al., 2012; Navarro and Fuss, 2009; Tuerlinckx et al., 2001).

Here, we present an R package, **RWiener**, that provides the four functions R uses to represent a distribution, implemented for the Wiener diffusion model: the *density function* `d` (Navarro and Fuss, 2009) to compute the probability density function (PDF) at a given quantile for a given parameter set; the *probability function* `p` (Blurton et al., 2012) to compute the cumulative distribution (CDF) at a given quantile for a given parameter set; the inverse CDF or *quantile function*¹ `q` to compute the quantile at a given p-value (CDF-value) for a given parameter set; and a *random number generator* (RNG) `r` (Tuerlinckx et al., 2001) to generate random samples for a given parameter set.

In addition, we extended the package to include several likelihood-based functions that can be used in combination with R's built-in estimation routines, like the `nlm` or `optim` function, to estimate the model parameters for an observed data set, assuming a Wiener diffusion process as the underlying process that created the data.

First, we will present the standard Wiener diffusion model with four parameters: the boundary separation α , the non-decision time τ , the bias parameter β and a drift rate parameter δ . Next, we will show the standard functionality of the **RWiener** package and how it can be used for parameter estimation. To conclude, we add a small discussion about the utility and extensibility of the here presented methods.

The Wiener diffusion model

For two-choice reaction time (2CRT) data coming from experiments in which subjects are asked to give one of two responses on a decision task, the Wiener diffusion model and its extensions have provided useful modeling approaches. In its simplest complete form, the Wiener diffusion model includes four parameters. The decision process is thought of as a continuous random walk, or diffusion, process that starts somewhere between two boundaries and ends as soon as it hits or crosses one or the other. The time of the first passage, and which boundary is hit first (and therefore which decision will be made) is probabilistic with a probability determined by the parameter set.

The standard Wiener diffusion model incorporates the following four parameters, a summary of which is provided in Table 1: (1) the boundary separation α which defines the distance between the two boundaries, with the first boundary being at 0 and the second being at α , (2) the non-decision time τ that defines the time that passes without any diffusion process going on (e.g., this incorporates the time needed to encode a stimulus and to execute a motor response), (3) the bias parameter β defining

¹Implemented using a reverse lookup algorithm of the CDF function.

Symbol	Parameter	Interpretation
α	Boundary separation	Speed-accuracy trade-off (high α means high accuracy)
β	Initial bias	Bias for either response ($\beta > 0.5$ means bias towards response 'A')
δ	Drift rate	Quality of the stimulus (close to 0 means ambiguous stimulus)
τ	Nondecision time	Motor response time, encoding time (high means slow encoding, execution)

Table 1: The four main parameters of the Wiener diffusion model, with their substantive interpretations. Reprinted with permission from Vandekerckhove (2009). Copyright 2009, Joachim Vandekerckhove and Department of Psychology and Educational Sciences, University of Leuven, Belgium.

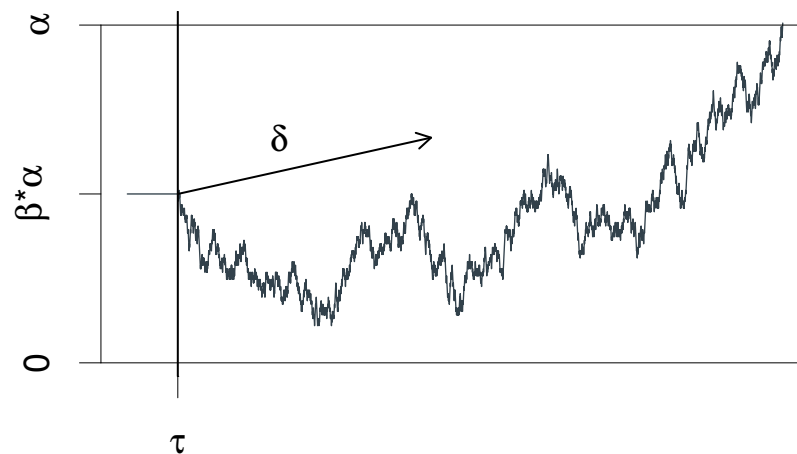


Figure 1: A graphical illustration of the Wiener diffusion model for two-choice reaction times. An evidence counter starts at value $\alpha\beta$ and evolves with random increments. The mean increment is δ . The process terminates as soon as the accrued evidence exceeds α or deceeds 0 . The decision process starts at time τ from the stimulus presentation and terminates at the reaction time.

the relative starting point of the diffusion process between the two boundaries (the absolute starting point ζ_0 can be obtained by $\zeta_0 = \beta\alpha$) and (4) the drift rate parameter δ which captures the tendency of the diffusion process to drift towards the upper boundary. Given these parameters, the latent information accumulation process is modeled with the stochastic process $\frac{d}{dt}X_t \sim \text{Normal}(\delta, s^2)$, with initial condition $X_0 = \alpha\beta$. The decision time is modeled as the first time at which $X_t \leq 0$ or $X_t \geq \alpha$, and the response time is the sum of the decision time and the non-decision time τ . Figure 1 shows an illustration of the Wiener diffusion model as described.

The parameters have the following restrictions: $0 < \beta < 1$, $\alpha > 0$, $\tau > 0$. Often, the variance parameter s is fixed at 0.1, whereas we fixed it at 1, yielding a more convenient interpretation of the drift rate parameter and greater computational efficiency. Conversion to a scale with the variance parameter fixed at any s can be done by multiplying the α and δ parameters by s .

The upper boundary may be defined as corresponding to correct trials, with the lower boundary corresponding to error trials. However, it often also makes sense to map qualitatively different responses to the two boundaries (e.g., in a same-different discrimination task, the upper bound could represent the 'different' responses). For generality, we will always speak simply of upper and lower boundaries. An overview of the computational strategy for the PDF and CDF functions is provided in the Appendix.

Usage of the package

The package can be obtained via CRAN for R version 2.15.0 or higher, and includes documentation that provides useful information and examples. As a guide through this demonstration of the **RWiener** package functionality, we use an example data set created by the random function of the package (using a RNG seed of 0 for exact reproducibility):

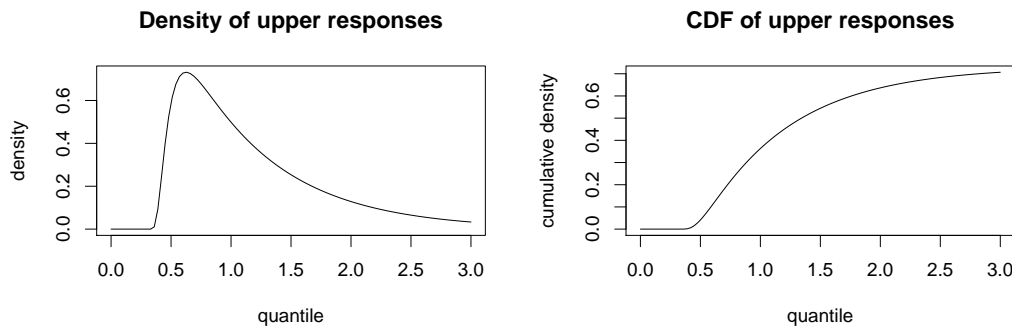


Figure 2: Example of plots made with function `dwiener` and `pwienier`.

```
set.seed(0)
dat <- rwiener(n=100, alpha=2, tau=.3, beta=.5, delta=.5)
```

The arguments for this function call are explained in the next paragraph. This command will generate a random data set, consisting of 100 observations, each generated from a random Wiener diffusion model process with the given four parameters. We used a drift rate parameter δ of 0.5, meaning that the drift diffusion process slightly tends towards the upper boundary. There was no initial bias towards either of the two boundaries, indicated by $\beta = 0.5$. The boundaries are separated by a value of $\alpha = 2$ and we used a non-decision time of $\tau = 0.3$, so there will never be a RT smaller than 0.3.

The created `dat` object is a dataframe with 100 observations and two variables: `dat$q` contains the RTs, whereas `dat$resp` contains a factor to indicate the response made (*upper* vs. *lower*). Note that the upper bound is assigned to the first level and the lower bound to the second one.

The four main functions of the package are described in the following subsection. All four functions are implemented in C, using the R library for C functions. These C functions are called inside end-user R functions, so that the end-user never calls the C functions directly, only indirectly through the provided R functions.

Standard distribution functions

To get the density of a specific quantile, one can use the density function `dwiener`:

```
dwiener(dat$q[1], alpha=2, tau=.3, beta=.5, delta=.5, resp=dat$resp[1], give_log=FALSE)
```

The function takes seven arguments, the first argument is the RT at which the density shall be evaluated, the next four arguments are the model parameters α , τ , β , and δ . The next argument, `resp` takes any of three valid values: *upper*, *lower*, or *both*, meaning that the function shall return the density at the given quantile for the upper boundary, the lower boundary or the sum of both densities together, respectively. Its default value is set to *upper*. Lastly, the function can be given an additional argument `give_log` that takes *TRUE* or *FALSE* as values and is set to *FALSE* as a default. When this argument is *TRUE*, the function will return the logarithm of the density instead of the density itself.

This function can also be used to draw simple plots. For example, the following code results in the plot in the left panel of Figure 2:

```
curve(dwiener(x, 2, .3, .5, .5, rep("upper", length(x))),
      xlim=c(0,3), main="Density of upper responses",
      ylab="density", xlab="quantile")
```

To calculate the cumulative distribution, we can use the CDF function `pwienier` as follows:

```
pwienier(dat$q[1], alpha=2, tau=.3, beta=.5, delta=.5, resp=dat$resp[1])
```

The arguments are the same as for the previously described function, without the `give_log` argument. Further, this function can be used in the same manner to draw plots, like the one shown in the right panel of Figure 2.

We can also find the appropriate quantile for a given probability via the inverse CDF function, implemented as `qwienier`:

```
# lookup of the .2 quantile for the CDF of the lower boundary
qwienier(p=.2, alpha=2, tau=.3, beta=.5, delta=.5, resp="lower")
```

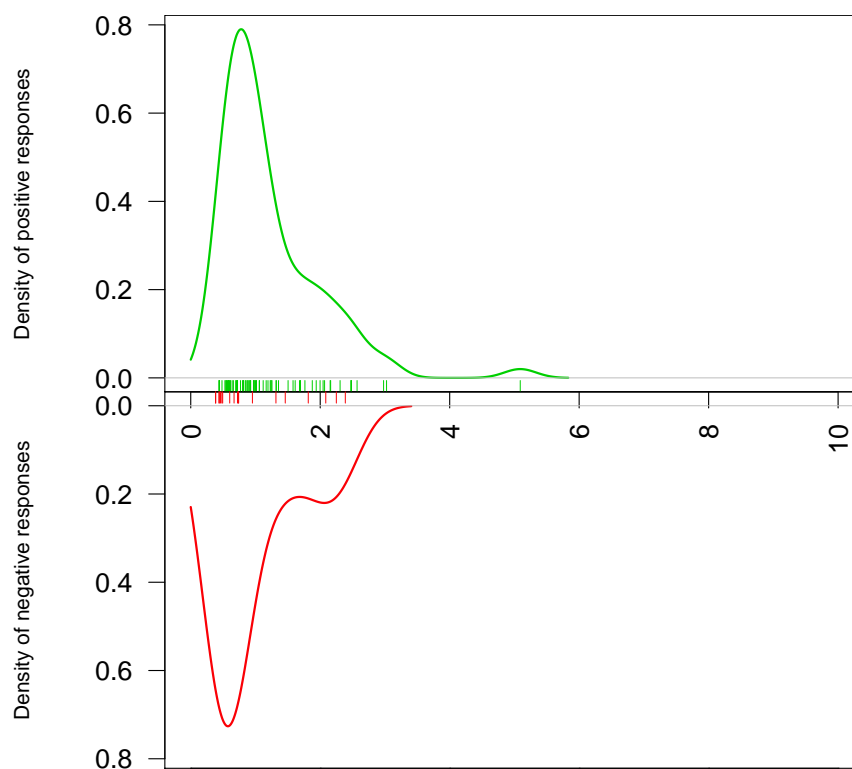


Figure 3: Plot drawn with `wiener_plot` function.

Again, the last five arguments of this function have the same meaning as for the two previously described functions. The first argument, however, is different: instead of a quantile, the first argument needs to be a probability, indicating the CDF-value to be looked up.

To round out the four standard distribution functions, the **RWiener** package further provides a function named `rwienr` that generates random values. Again, the last four input variables correspond to the model parameters and are the same as in the other distribution functions. The first argument `n` takes a number, indicating the desired number of randomly generated variates.

Plot function

To easily visualize data from the Wiener diffusion model, we can use the plot function `wiener_plot` as follows:

```
wiener_plot(dat)
```

This draws the two densities of the observed (or, in this case, generated) lower and upper responses. Figure 3 shows an example of this plot.

Parameter estimation with R's general-purpose optimization algorithms

In addition to the four distribution functions `dwiener`, `pwienr`, `qwienr`, `rwienr` and the plot function `wiener_plot`, the **RWiener** package provides four more functions designed for use in parameter estimation and model selection: (1) the `wiener_likelihood` function, which calculates the logarithm ℓ of the likelihood for given parameter values and data; (2) the `wiener_deviance` function, which calculates the model deviance -2ℓ ; (3) the `wiener_aic` function, which calculates the model AIC (Akaike's Information Criterion) $-2\ell + 8$; and (4) the `wiener_bic` function, which calculates the model BIC (Bayesian Information Criterion) $-2\ell + 4\ln(N)$.

```
x <- c(2, .3, .5, .5)
wiener_likelihood(x=x, dat=dat)
wiener_deviance(x=x, dat=dat)
wiener_aic(x=x, dat=dat)
wiener_bic(x=x, dat=dat)
```

The first argument of these functions, x , is a vector containing the four parameter values of the model in the order: $\alpha, \tau, \beta, \delta$. The second argument of these functions, dat , is a dataframe that contains the data points and the values of the two dependent variables: the RT and the boundary that was hit. This dataframe has to have the same shape as the dataframe dat as generated previously with the `rwien` function. The first column of the dataframe has to contain the RTs and the second column of the dataframe has to contain the response type (*upper* vs. *lower*).

Estimating parameters for a single condition

In order to estimate model parameters for a given data set, one can use R's `optim` or `nlm` function:²

```
# using optim, first with Nelder-Mead algorithm, then with BFGS
optim1 <- optim(c(1, .1, .1, 1), wiener_deviance, dat=dat, method="Nelder-Mead")
optim2 <- optim(optim1[["par"]], wiener_deviance, dat=dat, method="BFGS", hessian=TRUE)
```

```
# using nlm, which uses a Newton-type algorithm
nlm1 <- nlm(p=c(1, .1, .1, 1), f=wiener_deviance, dat=dat)
```

The obtained model parameter estimates can then be used to draw further inferences or create predictions. The help pages of the functions presented here show additional information and examples of usage.

Estimating parameters for multiple conditions

The deviance function can be combined to compute a single loss value for multiple conditions. To do so, first create a new inline function:

```
many_drifts <- function(x, datlist) {
  l = 0
  for (c in 1:length(datlist)) {
    l = l + wiener_deviance(x[c(1, 2, 3, c+3)], datlist[[c]])
  }
  return(l)
}
```

```
# create a second data set and a list containing both data sets
dat2 <- rwien(n=100, alpha=2, tau=.3, beta=.5, delta=1)
datlist <- list(dat, dat2)
```

```
# use nlm to estimate parameters
nlm1 <- nlm(p=c(1, .1, .1, 1, 1), f=many_drifts, dat=datlist)
```

If the input x is a parameter vector containing the first three parameters (α, τ, β) and then C drift rates, and dat is a list with C data sets, then the result will contain point estimates of three joint parameters in addition to C condition-specific drift rate estimates.

An alternative function can be defined that does not allow the drift rates to differ between conditions:

```
one_drift <- function(x, datlist) {
  l = 0
  for (c in 1:length(datlist)) {
    l = l + wiener_deviance(x, datlist[[c]])
  }
  return(l)
}
```

```
nlm2 <- nlm(p=c(1, .1, .1, 1), f=one_drift, dat=datlist)
```

Finally, the two competing models can be compared using the AIC criterion for model selection.³

```
AIC1 <- wiener_aic(x=nlm1$estimate, dat=datlist, loss=many_drifts)
AIC2 <- wiener_aic(x=nlm2$estimate, dat=datlist, loss=one_drift)
```

²Note that the `wiener_likelihood` function will return *Inf* for certain values, causing the `nlm` function to warn that the *Inf* was replaced by the maximum positive value.

³The input argument *loss*, with which a custom deviance function can be passed to compute the AIC for more complex models, is only available from **RWiener** v1.2.

For more details on the interpretation of the AIC criterion, see [Wagenmakers and Farrell \(2004\)](#).

Discussion

We presented **RWiener**, an R package that provides efficient algorithms to compute Wiener diffusion model functions and use these for further inference. This is the first package in R that provides full Wiener diffusion model functionality for R. We demonstrated how to use these functions and how to extend their usage to combine them with other R functions in order to do parameter estimation.

The presented functions can help researchers who work with 2CRT data to analyze their data in the Wiener diffusion model framework.

Authors' note

Correspondence concerning this article may be addressed to DW (dominik.wabersich@gmail.com) or JV (joachim@uci.edu). This project was supported by grant #1230118 from the National Science Foundation's Measurement, Methods, and Statistics panel to JV, and a travel grant from German Academic Exchange Service (PROMOS) to DW.

Bibliography

- S. Blurton, M. Kesselmeier, and M. Gondan. Fast and accurate calculations for cumulative first-passage time distributions in Wiener diffusion models. *Journal of Mathematical Psychology*, 56(6):470–475, 2012. [p49, 55, 56]
- D. J. Navarro and I. G. Fuss. Fast and accurate calculations for first-passage times in Wiener diffusion models. *Journal of Mathematical Psychology*, 53(4):222–230, 2009. [p49, 55]
- F. Tuerlinckx, E. Maris, R. Ratcliff, and P. De Boeck. A comparison of four methods for simulating the diffusion process. *Behavior Research Methods*, 33(4):443–456, 2001. [p49]
- J. Vandekerckhove. *Extensions and Applications of the Diffusion Model for Two-Choice Response Times*. PhD thesis, University of Leuven, 2009. [p50]
- J. Vandekerckhove and F. Tuerlinckx. Fitting the Ratcliff diffusion model to experimental data. *Psychonomic Bulletin & Review*, 14(6):1011–1026, 2007. [p49]
- J. Vandekerckhove and F. Tuerlinckx. Diffusion model analysis with MATLAB: A DMAT primer. *Behavior Research Methods*, 40(1):61–72, 2008. [p49]
- J. Vandekerckhove, F. Tuerlinckx, and M. Lee. Hierarchical diffusion models for two-choice response times. *Psychological Methods*, 16(1):44, 2011. [p49]
- A. Voss and J. Voss. fast-dm: A free program for efficient diffusion model analysis. *Behavior Research Methods*, 39(4):767–775, 2007. [p49]
- E.-J. Wagenmakers. Methodological and empirical developments for the Ratcliff diffusion model of response times and accuracy. *European Journal of Cognitive Psychology*, 21(5):641–671, 2009. [p49]
- E.-J. Wagenmakers and S. Farrell. AIC model selection using Akaike weights. *Psychonomic Bulletin & Review*, 11(1):192–196, 2004. [p54]
- E.-J. Wagenmakers, H. van der Maas, and R. Grasman. An EZ-diffusion model for response time and accuracy. *Psychonomic Bulletin & Review*, 14(1):3–22, 2007. [p49]
- T. V. Wiecki, I. Sofer, and M. J. Frank. HDDM: Hierarchical Bayesian estimation of the drift-diffusion model in Python. *Frontiers in Neuroinformatics*, 7(14), 2013. [p49]

Dominik Wabersich
 Department of Cognitive Sciences
 University of California, Irvine
 Irvine, CA 92697
 USA
dominik.wabersich@gmail.com

Joachim Vandekerckhove
 Department of Cognitive Sciences
 University of California, Irvine
 Irvine, CA 92697
 USA
joachim@uci.edu

Appendix

Mathematical descriptions of the density and distribution functions

Probability density function

The PDF of the diffusion model is that of the first-passage times of the accumulation process over the boundaries. That is, it is the expected distribution of the time until the process first hits or crosses one or the other boundary. This results in a bivariate distribution, over responses x and hitting times t . The PDF is approximated by Equation 1:

$$d(t, x = 0 | \alpha, \tau, \beta, \delta) = \frac{1}{\alpha^2} \exp \left[-\alpha\beta\delta - \frac{1}{2}\delta^2(t - \tau) \right] f \left(\frac{t - \tau}{\alpha^2} \middle| \beta \right). \quad (1)$$

The first passage time distribution for hits at the opposite boundary is $d(t, x = 1 | \alpha, \tau, \beta, \delta) = d(t, x = 0 | \alpha, 1 - \beta, \tau, -\delta)$. In Equation 1, f can be either the *large-time representation*

$$f_L(u | \beta) = \pi \sum_{k=1}^{+\infty} k \exp \left(-\frac{k^2 \pi^2 u}{2} \right) \sin(k\pi\beta)$$

or the *small-time representation*

$$f_S(u | \beta) = \frac{1}{\sqrt{2\pi u^3}} \sum_{k=-\infty}^{+\infty} (2k + \beta) \exp \left[-\frac{(2k + \beta)^2}{2u} \right]. \quad (2)$$

Either approximation of f can be more computationally efficient (in terms of the number of terms required to have the infinite sum converge below an error bound $\varepsilon = 10^{-10}$), depending on the parameter and data values. Specifically, Navarro and Fuss (2009) provide a decision rule to determine the more efficient formula: Equation 2 should be used if and only if

$$2 + \sqrt{-2u \log(2\varepsilon\sqrt{2\pi u})} - \sqrt{\frac{-2 \log(\pi u \varepsilon)}{\pi^2 u}} < 0,$$

where $u = \frac{t - \tau}{\alpha^2}$. The derivation of this decision rule, as well as demonstrations of its efficiency, are given in Navarro and Fuss (2009).

Cumulative distribution function

A similar rule for the efficient computation of the CDF can be found in Blurton et al. (2012)—there exists a large-time representation

$$\begin{aligned} p_L(t, x = 0 | \alpha, \tau, \beta, \delta) &= P_0 - \frac{2\pi}{\alpha^2} \exp \left(-\alpha\beta\delta - \frac{\delta^2(t - \tau)}{2} \right) \\ &\times \sum_{k=1}^{+\infty} \frac{k \sin(\pi k\beta)}{\delta^2 + (k\pi/\alpha)^2} \exp \left[-\frac{1}{2} \left(\frac{k\pi}{\alpha} \right)^2 (t - \tau) \right], \end{aligned} \quad (3)$$

and a small-time representation

$$\begin{aligned} p_S(t, x = 0 | \alpha, \tau, \beta, \delta) &= P_0 - \operatorname{sgn} \delta \sum_{k=-\infty}^{+\infty} \left\{ \exp(-2\delta\alpha k - 2\delta\alpha\beta) \Phi \left[\operatorname{sgn} \delta \frac{2\alpha k + \alpha\beta - \delta(t - \tau)}{\sqrt{t - \tau}} \right] \right. \\ &\quad \left. - \exp(2\delta\alpha k) \Phi \left[\operatorname{sgn} \delta \frac{-2\alpha k - \alpha\beta - \delta(t - \tau)}{\sqrt{t - \tau}} \right] \right\}, \end{aligned} \quad (4)$$

where sgn is the signum function and Φ denotes the cumulative normal distribution function. In both cases, P_0 is the probability of a hit at the lower boundary,

$$P_0 = \begin{cases} \frac{1 - \exp[-2\delta\alpha(1-\beta)]}{\exp(2\delta\alpha\beta) - \exp[-2\delta\alpha(1-\beta)]} & \text{if } \delta \neq 0, \\ 1 - \beta & \text{if } \delta = 0. \end{cases}$$

In order to reach an approximation with absolute error not exceeding $\varepsilon = 10^{-10}$, the infinite sums in Equations 3 and 4 are approximated with K_L and K_S terms, respectively (derivation of these limits is given in [Blurton et al., 2012](#)). K_L and K_S are the smallest integers that satisfy the following constraints:

$$\begin{cases} K_L^2 & \geq \frac{1}{t-\tau} \left(\frac{\alpha}{\pi}\right)^2 \\ K_L^2 & \geq -\frac{2}{t-\tau} \left(\frac{\alpha}{\pi}\right)^2 \left\{ \log \left[\frac{1}{2} \varepsilon \pi (t-\tau) \left(\delta^2 + \frac{\pi^2}{\alpha^2} \right) \right] + \delta\alpha\beta + \frac{1}{2} \delta^2 (t-\tau) \right\} \end{cases}$$

and

$$\begin{cases} K_S & \geq \beta - 1 + \frac{1}{2\delta\alpha} \log \left\{ \frac{\varepsilon}{2} [1 - \exp(2\delta\alpha)] \right\} \\ K_S & \geq \frac{1}{2\alpha} \left[0.535 \sqrt{2(t-\tau)} + \delta(t-\tau) + \alpha\beta \right] \\ K_S & \geq \frac{1}{2}\beta - \frac{1}{2\alpha} \sqrt{t-\tau} \Phi^{-1} \left\{ \frac{\varepsilon\alpha}{0.3\sqrt{2\pi(t-\tau)}} \exp \left[\frac{1}{2} \delta^2 (t-\tau) + \delta\alpha\beta \right] \right\} \end{cases}.$$

Due to the larger computational effort of the small-time representation of the CDF, the large-time representation is selected if $K_L/K_S < 10$.

PivotalR: A Package for Machine Learning on Big Data

by Hai Qian

Abstract **PivotalR** is an R package that provides a front-end to PostgreSQL and all PostgreSQL-like databases such as Pivotal Inc.'s Greenplum Database (GPDB), HAWQ. When running on the products of Pivotal Inc., **PivotalR** utilizes the full power of parallel computation and distributive storage, and thus gives the normal R user access to big data. **PivotalR** also provides an R wrapper for MADlib. MADlib is an open-source library for scalable in-database analytics. It provides data-parallel implementations of mathematical, statistical and machine-learning algorithms for structured and unstructured data. Thus **PivotalR** also enables the user to apply machine learning algorithms on big data.

Introduction

In recent years, Big Data has become an important research topic and a very realistic problem in industry. The amount of data that we need to process is exploding, and the ability of analyzing big data has become the key factor in competition. Big data sets do not fit into computer's memory and it would be really slow if the big data sets were processed sequentially. On the other hand, most contributed packages of R are still strictly sequential, single machine, and they are restricted to small data sets that can be loaded into memory. As computing shifts irreversibly to parallel architectures and big data, there is a risk for the R community to become irrelevant.

Some efforts have been invested into developing packages that give R users and developers the access to some of the parallel distributive platforms. Some examples include **dplyr** (Wickham and Francois, 2013), **RHadoop** (Revolution Analytics Company, 2012) (including **plyrmr**, **rnr**, **rhdfs**, **rhbase**), **RHIPE** (Guha et al., 2012), **RHive** (NexR, 2013), **teradataR** (Teradata Corporation, 2012) etc.

In this paper we introduce the package **PivotalR** (Pivotal Inc., 2013d), which provides an R front-end with data.frame oriented API for R users to access big data stored in distributive databases or Hadoop distributive file system (HDFS) (The Apache Software Foundation, 2013). In this sense, **PivotalR** is close to the still under development **dplyr** package, which has a data.frame oriented API and multiple back-ends including several SQL database systems. **PivotalR** puts more emphasis on machine learning by providing a wrapper for MADlib (Pivotal Inc., 2013c), which is an open-source library of scalable in-database machine learning algorithms. Actually **PivotalR** offers more than what MADlib has. It adds functionalities that do not exist in MADlib, for example, the support for categorical variables. **PivotalR** makes it easier to work on big data sets in databases.

This package is especially useful for users who are not familiar with SQL language, because the functions and syntax for the manipulation of tables are very similar to those for data.frame manipulation defined natively in R. Thus the learning curve for the package is very smooth. **PivotalR** is also very useful for users who are familiar with SQL language, because it brings the graphical and analytical functionalities of R to the processing of big data stored in databases.

PivotalR is contributed to the open-source community by the Predictive Analytics Team at Pivotal Inc. In order to gain the power of distributive storage and parallel computation, the users need to have a distributive database system installed, for example the Greenplum Database or HAWQ. **PivotalR** also supports the open-source database system PostgreSQL. Therefore **PivotalR** benefits R users by providing an easy-to-access interface for manipulating data tables stored in PostgreSQL database system and a combination of the powers of R and MADlib, which lets the user directly apply machine learning algorithms on the big data stored in databases.

It is worth mentioning that **PivotalR** provides the access to data stored on HDFS by supporting HAWQ (Pivotal Inc., 2013b), which is the SQL query engine on HDFS created by Pivotal Inc.

The user does not need to worry about the restriction of memory size even if the data size is very big, because **PivotalR** minimizes the amount of data transferred between the database and R. The user manipulates the data from R but the data itself stays in the database.

The work flow of using **PivotalR** is the following. First, the user uses "db.connect" to connect to a database. Then, "db.data.frame" can be used to create a wrapper for a data table in the database. Minimal information about the table is kept in the wrapper, and no data is loaded into the memory. The user can easily operate on the "db.data.frame" wrapper object and any operation creates a "db.Rquery" object, which is just a wrapper for a series of operations and contains a SQL query string (see the section of Manipulation of Data Tables in Database). During this data preparation step, no

data is loaded. In the next step, the user can either call the "lookat" function to view a sample of the operation result, or call one of the MADlib wrapper functions to execute a machine learning algorithm. Both choices initiate computation in the connected database. Usually the computation result is small and can be loaded into memory for further processing. In some cases, the result is also big, for example, the MADlib wrapper for ARIMA produces the residuals for each row of the data. In such cases, a table is created in the database to store the result and a wrapper is automatically created in R, and the user can use the same work flow to analyze the result table.

From the above work flow, one can see that all the actual computation is done in the database. So the performance depends on the database. For the Greenplum database or HAWQ, all computation is done in a parallel and distributive way. Especially the MADlib machine learning functions are all implemented to fully utilize the parallel and distributive power of the databases. There are some operations that cannot be done in a parallel way, for example the ordered aggregate, but they have not been implemented in **PivotalR** yet. For the Postgres database, even though the computation is not in parallel, the user can still use PivotalR to easily access the data stored in the database, which might be too big to load into memory. Actually the performance on Postgres is not bad at all. In our performance tests in Postgres database, PivotalR's ARIMA wrapper function for MADlib is faster than R's own "arima" function when applied on large data sets.

It is worth mentioning that the Greenplum database (GPDB) (Pivotal Inc., 2013a) has a community edition, which is free (but not open-source) and has all the functionalities except the technical support from Pivotal Inc.

At the time of writing, the version of **PivotalR** on CRAN is 0.1.8.

In this paper, we briefly introduce the usage of **PivotalR**. First in the next section we explain the architecture of **PivotalR**. Then we use various examples to illustrate the usage and work flow of **PivotalR**.

Architecture of PivotalR

As is shown in Fig. 1, **PivotalR** connects to the database through the **RPostgreSQL** (Conway et al., 2013) package. However, **PivotalR** does not directly call the functions in **RPostgreSQL**. Instead we create an abstraction layer to wrap the functions of **RPostgreSQL**, and the access to the database is done by calling the functions in the abstraction layer, which then call the functions in **RPostgreSQL**. This design makes it easier to add supports for other data storage platforms in the future. Right now, **PivotalR** only supports the connection to PostgreSQL, GPDB (Greenplum Database) and HAWQ. Through HAWQ, **PivotalR** can access the big data stored on HDFS.

In the future, **PivotalR** will add supports to other DBMS, Pivotal HD (Hadoop created by Pivotal Inc.) and Hadoop (MapReduce), but we have not decided about how to do that. This is why there are question marks in Fig. 1.

The table operation functions are built upon the abstraction layer for accessing data. The syntax of these functions is the same as R's "data.frame" operation functions, such as [, [<-, \$, \$<-, merge, "by" etc. Arithmetic methods such as +, -, *, /, logical functions such as >, <, ==, !=, <=, >=, !, |, &, math functions such as log, exp, factorial etc. are all implemented.

The layers that are built upon the table operations are the MADlib function wrappers and other functions, which will be covered in the subsequent sections.

A graphical user interface using the **shiny** (RStudio Inc., 2013) package is also implemented. Right now it only provides the graphical interface to the MADlib wrapper functions, but we have the plan to provide the graphical access to all functionalities of **PivotalR** in the future.

It is worth explaining more about how **PivotalR** operates on the tables. Before everything, the user needs to create a "db.data.frame" object, which is just a wrapper of the table. Any operation on this object is translated into an SQL query string. However, the SQL query is not executed at this point, and is instead stored in an object of class "db.Rquery" in R. When the user wants to execute the operation, he can call the function "preview" or "lookat" (or a shorthand "lk") to execute the operation and load part or all of the results into memory to view. The user can also choose to use "as.db.data.frame" to execute the SQL query and save the result into a new table. This design gives the user the freedom to choose when to load data into memory or create tables to store intermediate data in the process of a calculation. It also avoids the risk of accidentally loading big data into memory.

In order to realize the above design, we use a class hierarchy shown in Fig. 2. **PivotalR** uses S4 object-oriented programming extensively. All objects that are related to the data in the database belong to subclasses of an abstract class named "db.object".

As has been mentioned, the class "db.data.frame" is a wrapper of data objects stored in the connected database. When it is created by the command "db.data.frame", no actual data stored in

object. Essentially, a "db.Rquery" object is just a container of a string of SQL query, which can be viewed using the command "content". The command "as.db.data.frame" creates a table in the database using the results of the SQL query contained in a "db.Rquery" object and then creates a "db.data.frame" wrapper that points to this table.

"preview" (or its alias "lookat" and "lk") transfers the data from the database into R's memory. By default it fetches 100 rows of data from the table in the database and converts the data into a "data.frame" in R.

```
> lookat (x, 10)
  id sex length diameter height whole shucked viscera shell rings
1   4  M  0.440    0.365  0.125 0.5160  0.2155  0.1140 0.155   10
2   8  F  0.545    0.425  0.125 0.7680  0.2940  0.1495 0.260   16
3  12  M  0.430    0.350  0.110 0.4060  0.1675  0.0810 0.135   10
4  16  M  0.500    0.400  0.130 0.6645  0.2580  0.1330 0.240   12
5  20  M  0.450    0.320  0.100 0.3810  0.1705  0.0750 0.115    9
6  24  F  0.550    0.415  0.135 0.7635  0.3180  0.2100 0.200    9
7  28  M  0.590    0.445  0.140 0.9310  0.3560  0.2340 0.280   12
8  32  F  0.680    0.560  0.165 1.6390  0.6055  0.2805 0.460   15
9  36  M  0.465    0.355  0.105 0.4795  0.2270  0.1240 0.125    8
10 40  M  0.355    0.290  0.090 0.3275  0.1340  0.0860 0.090    9
```

The MADlib wrappers can be applied to both "db.data.frame" and "db.Rquery" objects. When a MADlib wrapper function is applied to a "db.Rquery" object, a temporary table is created using as.db.data.frame. MADlib is an in-database library and we have to convert "db.Rquery" into an object inside the database so that we could apply MADlib functions to it. This temporary table is dropped after the computation is done.

The design of **PivotalR** minimizes the amount of data that needs to be transferred between the database and R. The only function that transfers data from the database is "preview".

Manipulation of data tables in database

As has been mentioned, **PivotalR** overloads many methods that operate on "data.frame", and thus lets the operations on the "db.obj" objects mimic those on "data.frame" as much as possible, with one important difference discussed below.

```
> x <- db.data.frame ("abalone")
> dim (x)
[1] 4177  10
> names (x)
[1] "id"      "sex"      "length"   "diameter" "height"   "whole"
[7] "shucked" "viscera"  "shell"    "rings"
> x$rings <- x$rings + 2 / 3
> x$newCol <- (x$rings + x[['length']]) < 10
> content(x)
[1] "select \"id\" as \"id\", \"sex\" as \"sex\", \"length\" as \"length\",
\"diameter\" as \"diameter\", \"height\" as \"height\", \"whole\" as \"whole\",
\"shucked\" as \"shucked\", \"viscera\" as \"viscera\", \"shell\" as \"shell\",
(\"rings\")::double precision + 0.666666666666667 as \"rings\",
(((\"rings\")::double precision + 0.666666666666667)::double precision
+ (\"length\")::double precision < 10 as \"newCol\" from \"abalone\""
```

Here "content" displays the SQL query stored in the "db.Rquery". The quotes inside the SQL query is used to deal with column or table names with special characters. x was originally a "db.data.frame" object, but was converted to a "db.Rquery" object by the operation x\$rings <- x\$rings + 2 / 3.

As long as a "key" is specified when the user calls "db.data.frame" function, the syntax A[3,4] can be used. Here the first number "3" means that the key column has the value of 3. Therefore, lk(A[3,4],-1) may return multiple values. This is different from the behavior of "data.frame", where A[3,4] returns the element on 3-rd row and 4-th column. This difference is related to the lack of intrinsic order in the database tables, which will be discussed in the following.

The rows of a "data.frame" object have an intrinsic order, which makes the operations that involve two "data.frame" objects easier to define. For example, it is straightforward to define the subtraction of two columns of two different "data.frame" objects by matching the order of the rows as long as they have the same number of rows.

On the other hand, the tables in the database do not have an intrinsic order for the rows. Thus, most operations like subtraction, addition etc. can only be defined for columns belonging to the same table. However, this is usually not a problem. Whenever the user wants to do an operation that involves two tables, he/she should first call the function "merge" to create a new table (actually a "db.Rquery" object) from the two tables and at the same time explicitly specify how to match the rows of the two different tables. Then, the user can easily do operations on the new table. While R's "data.frame" implicitly uses the intrinsic order of the rows to match the rows from two different data.frames, this needs to be explicitly done for "db.data.frame" objects. For example,

```
> x <- db.data.frame("abalone")
> y <- db.data.frame("abalone")
> z <- merge(x, y, by = NULL) # cross join two tables
> names(z)
[1] "id_x"      "sex_x"      "length_x"   "diameter_x" "height_x"
[6] "whole_x"   "shucked_x"  "viscera_x"  "shell_x"    "rings_x"
[11] "id_y"      "sex_y"      "length_y"   "diameter_y" "height_y"
[16] "whole_y"   "shucked_y"  "viscera_y"  "shell_y"    "rings_y"
> z$sex_x == z$sex_y # This is a db.Rquery object
```

The data frame 'abalone' is lazy-loaded in **PivotalR**. It is used as an example data here. One can use `as.db.data.frame` to transfer this data set into a table in the database and at the same time create a wrapper for the table.

```
> x <- as.db.data.frame(abalone, conn.id=1)
The data in the data.frame abalone is stored into the table in
database madlib on localhost !
```

All the subsequent operations will be applied on the table wrapper `x`. A sample of the data is shown in the following, where `sort` is used to create a sorting operation on the original table.

```
> lookat(sort(x, FALSE, x$id), 10) # load 10 rows of the sorted data
  id sex length diameter height whole shucked viscera shell rings
1   1  M  0.455    0.365  0.095 0.5140  0.2245  0.1010 0.150   15
2   2  M  0.350    0.265  0.090 0.2255  0.0995  0.0485 0.070    7
3   3  F  0.530    0.420  0.135 0.6770  0.2565  0.1415 0.210    9
4   4  M  0.440    0.365  0.125 0.5160  0.2155  0.1140 0.155   10
5   5  I  0.330    0.255  0.080 0.2050  0.0895  0.0395 0.055    7
6   6  I  0.425    0.300  0.095 0.3515  0.1410  0.0775 0.120    8
7   7  F  0.530    0.415  0.150 0.7775  0.2370  0.1415 0.330   20
8   8  F  0.545    0.425  0.125 0.7680  0.2940  0.1495 0.260   16
9   9  M  0.475    0.370  0.125 0.5095  0.2165  0.1125 0.165    9
10 10  F  0.550    0.440  0.150 0.8945  0.3145  0.1510 0.320   19
```

Finally, I want to reiterate that all these operations done on "db.data.frame" are not executed immediately. Instead a "db.Rquery" object is produced. The user can convert it into a real table using `as.db.data.frame`.

MADlib wrapper functions

In the next, we show some examples of MADlib wrapper functions. As is shown in Fig. 2, all MADlib wrapper functions can be applied to "db.obj" objects. Because the connection information is already contained in "db.obj" objects, we do not need to specify "conn.id" in the MADlib wrapper functions.

First, we show several examples of linear regression. Basically, **PivotalR**'s "madlib.lm" uses the same formula syntax as "lm". Because MADlib's linear regression supports fitting separate linear regression models on sub-groups of the data, which are grouped by one or multiple columns, "madlib.lm" allows "|" in the formula, which means that the model is fit on a subset of the data conditioned on the value of the variables after "|".

```
> ## fit one different model to each group of data with the same sex
> fit <- madlib.lm(rings ~ . - id | sex, data = x)
```

sex has 3 distinct values, and thus fit is a list of 3 elements. Each element is the linear regression model for a subset of the data. The mean square error can be easily computed

```
> ## apply the model to data in another database
> lookat(mean((x$rings - predict(fit, x))^2)) # mean square error
```



```
rings_madlib_predict_opr_opr_avg
1 4.647291
```

We can plot a random sample of the fitted values together with the real values, as is shown in Fig. 3. We can also select one model from the fitting result to plot, as is shown in the comments of the following snippets.

```
> ## plot the result
> ap <- cbind(x$rings, predict(fit, x)) # combine two columns
> plot(lookat(sort(ap, FALSE, NULL), 100)) # plot a random sample
> ## ap <- cbind(x$rings[x$sex == "I"], predict(fit[[1]], x[x$sex == "I",]))
> ## plot(lookat(sort(ap, FALSE, NULL), 100)) # plot a random sample
```

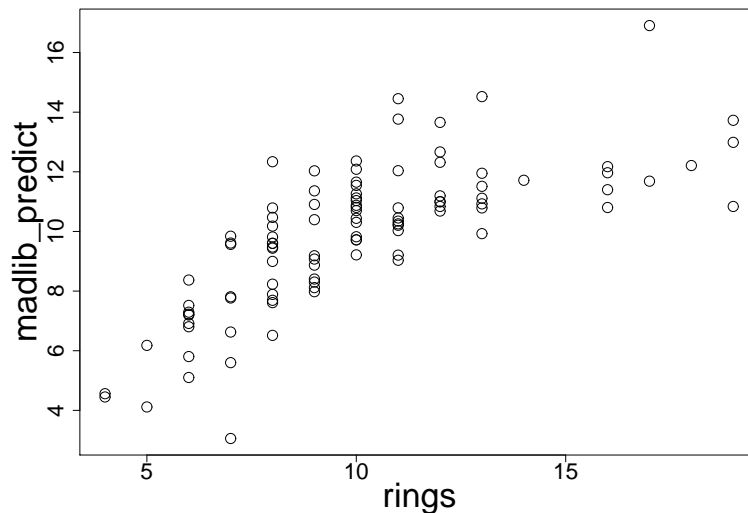


Figure 3: The sample plot

Support for categorical variables is done through "as.factor", as is shown in the next example.

```
> v <- x
> v$sex <- as.factor(v$sex) # specify which column to pivot
> f <- madlib.lm(rings ~ . - id, data = v)
> f
MADlib Linear Regression Result

Call:
madlib.lm(formula = rings ~ . - id, data = v)

-----

Coefficients:
(Intercept)  3.89464  0.29157 13.3576  6.992e-40 ***
sex:M        0.05772  0.08335  0.6925  4.887e-01
sex:I       -0.82488  0.10240 -8.0558  1.022e-15 ***
length     -0.45834  1.80912 -0.2533  8.000e-01
diameter    11.07510  2.22728  4.9725  6.876e-07 ***
height     10.76154  1.53620  7.0053  2.861e-12 ***
whole       8.97544  0.72540 12.3730  1.469e-34 ***
shucked    -19.78687  0.81735 -24.2086  2.988e-121 ***
viscera    -10.58183  1.29375 -8.1792  3.757e-16 ***
shell       8.74181  1.12473  7.7723  9.639e-15 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-squared: 0.5378844
Condition Number: 137.2694
```

Because `extractAIC` methods are implemented for the result of `madlib.lm` and `madlib.glm`, one can use R's `step` function for feature selection.

```
fit <- madlib.glm(rings < 10 ~ ., data = x, family = "binomial")
step(fit)
```

This is a good example of how R's analytical functionalities can be combined with MADlib.

In the next example, we use the bootstrap aggregate method to fit a linear model to the data.

```
## generic bagging
fit <- generic.bagging(function(data) {
  madlib.lm(rings ~ . - id - sex, data = data)
},
data = x, nbags = 10, fraction = 0.85)

pred <- predict(fit, newdata = x) # make prediction

lookat(mean((x$rings - pred)^2))
```

Just like "glm" in the **stats** package, **PivotalR** provides "madlib.glm", which can do both linear and logistic regressions. The usage is similar to `glm`. Please refer to **PivotalR** manual ([Pivotal Inc., 2013d](#)) for details.

Support for array columns

Both Greenplum and Postgres databases have an upper limit for the number of columns in a data table. The upper limit depends on the column data types, but is usually around 1600. If the big data set has more features, the only workaround is to use an array column to encapsulate all the features, because there is no limit for how many elements an array can contain.

PivotalR has full support for such array columns, as is shown in the next example.

```
> z <- db.data.frame("madlibtestdata.lin_auto_mpg_oi")
> lookat(z, 10)
  x.1 x.2 x.3 x.4 x.5 x.6 x.7 y
1   8   8 383 170 3563 10.0 70 1 15
2   8   8 340 160 3609 8.0 70 1 14
3   8   8 400 150 3761 9.5 70 1 15
4   4  121 113 2234 12.5 70 2 26
5   8   8 360 215 4615 14.0 70 1 10
6   8  304 193 4732 18.5 70 1 9
7   4  113 95 2228 14.0 71 3 25
8   6  250 88 3302 15.5 71 1 19
9   8  351 153 4154 13.5 71 1 14
10  8  318 150 4096 13.0 71 1 14
> madlib.lm(y ~ x, data = z)
MADlib Linear Regression Result
```

```
Call:
madlib.lm(formula = y ~ x, data = z)
```

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-17.218435	4.644294	-3.7074	2.402e-04 ***
x[1]	-0.493376	0.323282	-1.5261	1.278e-01
x[2]	0.019896	0.007515	2.6474	8.445e-03 **
x[3]	-0.016951	0.013787	-1.2295	2.196e-01
x[4]	-0.006474	0.000652	-9.9288	7.883e-21 ***
x[5]	0.080576	0.098845	0.8152	4.155e-01
x[6]	0.750773	0.050973	14.7288	3.069e-39 ***
x[7]	1.426140	0.278136	5.1275	4.666e-07 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-squared: 0.8214781
Condition Number: 85850.33

Handling of NULL values

PivotalR makes it very easy to filter NULL values from a table. Here we show an example using the data "null.data" that comes with **PivotalR**. "null.data" has lots of NULL value. However, we can easily filter out all the NULL values.

```
> delete("null_data", conn.id = 2)
[1] TRUE
> w <- as.db.data.frame(null.data, "null_data", conn.id = 2)
The data in the data.frame null.data is stored into the table null_data in database
madlib on localhost !
> dim(w)
[1] 67126    10
> lookat(w, 10)
      sf_mrtg_pct_assets ris_asset lncrcd lnauto lnconoth lnconrp intmsrfv
1             31.62    208596      0     NA    18260      44         0
2             20.66    34647      0     NA     2997       0         0
3             14.95   160175      0     NA    15164       0         0
4             17.49   117398      0     NA    12365       0         0
5             35.59   233592      0     NA     6261     151         0
6             11.57   166387      0     NA    36356       0         0
7             10.56   102891      0     NA     6311       0         0
8              6.05    30138      0     NA     2664       0         0
9             14.53  1615727      0     NA    80829    1106        105
10            12.88    83090      0     NA    11815       0         0
      lnrenr1a lnrenr2a lnrenr3a
1           NA       NA       NA
2           NA       NA       NA
3           NA       NA       NA
4           NA       NA       NA
5           NA       NA       NA
6           NA       NA       NA
7           NA       NA       NA
8           NA       NA       NA
9           NA       NA       NA
10          NA       NA       NA
> db.objects("null", conn.id = 2)
[1] "madlibtestdata.credit_nulls"      "madlibtestdata.rf_golf_nullclass"
[3] "madlibtestdata.rf_nursery_nullclass" "madlibtestdata.table_has_null"
[5] "public.null_data"
> for (i in 1:10) w <- w[!is.na(w[i]),] # filter NULL values
> dim(w)
[1] 6789    10
> madlib.lm(sf_mrtg_pct_assets ~ ., data = w)
MADlib Linear Regression Result
```

Call:
madlib.lm(formula = sf_mrtg_pct_assets ~ ., data = w)

```
-----
Coefficients:
      Estimate Std. Error  t value    Pr(>|t|)
(Intercept)  1.528e+01  1.471e-01 103.90694 0.000e+00 ***
ris_asset    1.602e-08  1.141e-08   1.40440 1.602e-01
lncrcd      -1.608e-07  7.725e-08  -2.08170 3.741e-02 *
lnauto      -3.871e-07  3.447e-07  -1.12301 2.615e-01
lnconoth     -7.498e-07  4.370e-07  -1.71559 8.628e-02 .
lnconrp       7.575e-08  1.229e-06   0.06165 9.508e-01
```

```

intmsrfv      8.609e-06  1.790e-06  4.81005 1.542e-06 ***
lnrenr1a     -4.827e-06  3.837e-05  -0.12580 8.999e-01
lnrenr2a      1.358e-04  2.359e-05  5.75605 8.986e-09 ***
lnrenr3a     -3.133e-05  4.273e-06  -7.33355 2.502e-13 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-squared:  0.01083685
Condition Number: 37854014

```

na.action is supported by madlib.lm, madlib.glm and madlib.elnet. The user can define his own "na.action" function. "na.omit" method is also implemented.

Quickly creating prototypes of machine learning algorithms

One of the goals of **PivotalR** is to help data scientists quickly create prototypes for parallel machine learning algorithms that can run in distributive databases. Although **PivotalR** is still at an early stage of development, we have already invested some effort into this. The idea is to implement some basic operations that forms the building blocks for most machine learning algorithms.

For example, we implemented the function `crossprod`, which computes the operation $X^T Y$. X and Y are wrappers created by "db.data.frame" for tables that represent matrices. X has m rows (observations) and n columns (features). The assumption here is that m can be very large but the number of features n is small. Thus results of $X^T X$ and $X^T Y$ are both small and can be loaded into memory for further calculations. The big data computation only involves the computations of $X^T X$ and $X^T Y$ in the database.

```

## linear regression
linregr <- function (x, y)
{
  a <- crossprod (x)
  b <- crossprod (x, y)
  solve (lookat (a)) %*% lookat(b)
}

linregr (db.array(1, x[, -c (1,2,10)], x[["rings"]]))

```

Another example is the implementation of PCA for a matrix Z . Again we assume that the number of columns n is not large and the product $Z^T Z$ is small enough to be loaded into memory. **PivotalR** implements "scale", which is used together with `crossprod` in this example.

```

## PCA
## compute all eigenvectors in parallel
## can be used for tables with features < 1000

pca <- function (x, center = TRUE, scale = FALSE)
{
  y <- scale(x, center = center, scale = scale) # centering and scaling
  z <- as.db.data.frame(y, verbose = FALSE) # create an intermediate
  # table to speed up computation
  m <- lookat(crossprod(z)) # one scan of the table to compute Z^T * Z
  d <- delete(z) # delete the intermediate table
  res <- eigen(m) # only this computation is in R
  n <- attr(y, "row.number") # save the computation to count rows

  ## return the result
  list(val = sqrt(res$values/(n-1)), # eigenvalues
       vec = res$vectors, # columns of this matrix are eigenvectors
       center = attr(y, "scaled:center"),
       scale = attr(y, "scaled:scale"))
}

dat <- db.data.frame("madlibtestdata.pca_mat_600_100", conn.id = 2)

q <- pca(dat[, -1])

```

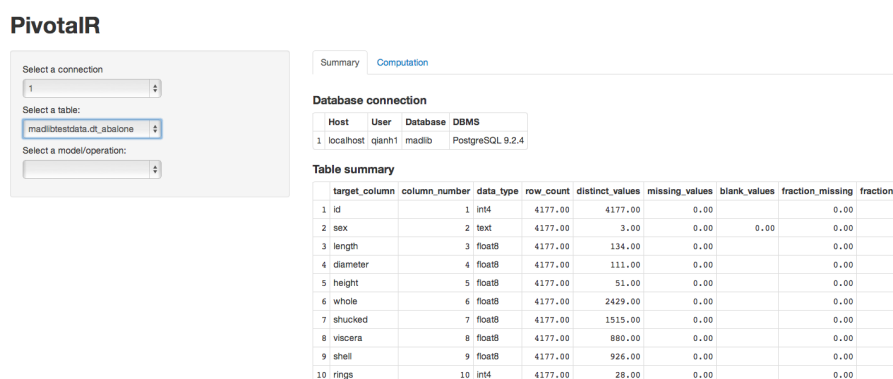


Figure 4: Display the summary of a table in the graphical interface.

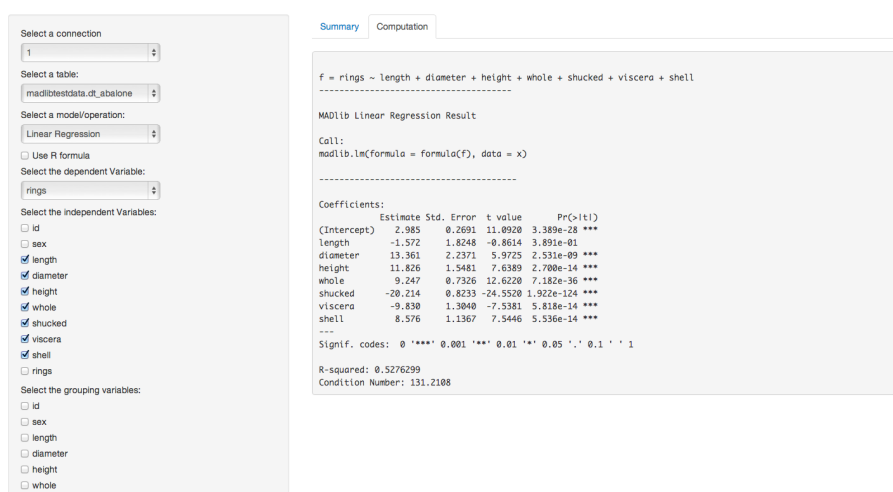


Figure 5: Display the result of linear regression in the graphical interface.

GUI of PivotalR

As a front-end to both the database and MADlib, **PivotalR** also provides a graphical user interface (GUI) to let users that are not familiar with data science have an easy access to the functionalities. The GUI is written using the package **shiny**, which provides a very nice web application framework to create graphical user interface.

The GUI is launched by the command `pivotalr` after the connection to the databases have been created. Fig. 4 and Fig. 5 show the graphical user interface. Fig. 4 shows a screenshot for the summary of various properties for all columns of a table. The actual computation is done through MADlib's "summary" function, which has a wrapper function "madlib.summary" in **PivotalR**. Fig. 5 shows the computation result of linear regression. The actual computation is done through MADlib's `linreg_train` function, which has a wrapper function "madlib.lm".

The GUI of **PivotalR** is based upon the package **shiny**, which is a web application framework. Therefore the command `pivotalr` launches a web application. One can run the GUI on a server and connect to it using a web browser.

The GUI of **PivotalR** is still at its early development stage. Many functionalities, especially the various operations on data tables, cannot be done through GUI. However, we expect to improve the GUI gradually in the future releases.

Future work

MADlib has a wide variety of machine learning functions. At the time of writing, the version of **PivotalR** on CRAN is 0.1.8, and it implements 5 wrapper functions: linear regression, logistic regression, ARIMA, elastic net regularization, and the data table summary function. In the future, all MADlib functions will have their wrappers in **PivotalR**. This is our highest priority right now.

As to the shiny interface of **PivotalR**, we are still exploring the possibilities of streamlining the machine learning workflow and making it compatible with a graphical interface. This by itself is a big project, but it has a lower priority in our road-map.

The support for other platforms, like Hadoop or other database systems, is also on our road-map, but has a relatively low priority too.

Summary

Here we introduced **PivotalR**, a package created by Pivotal Inc. This package provides an R front-end to PostgreSQL and all PostgreSQL-like databases like Pivotal Inc.'s Greenplum Database (GPDB), HAWQ. **PivotalR** allows the user to manipulate tables in database from within R. It also provides the wrapper functions for MADlib, which is an in-database machine learning library. **PivotalR** implements the support for array columns and NULL handling. It also helps the user to quickly create prototypes of machine learning algorithms. A graphical user interface based on shiny is also implemented. In general, **PivotalR** gives the user access to big data stored in databases and an easy-to-use machine learning toolkit.

Bibliography

- J. Conway, D. Eddelbuettel, T. Nishiyama, S. K. Prayaga, and N. Tiffin. *RPostgreSQL: R interface to the PostgreSQL database system*, 2013. URL <http://CRAN.R-project.org/package=RPostgreSQL>. R package version 0.4. [p58]
- S. Guha, R. Hafen, J. Rounds, J. Xia, J. Li, B. Xi, and W. S. Cleveland. Large complex data: divide and recombine (d&r) with rhipe. *Stat*, 1(1):53–67, 2012. [p57]
- NexR. *RHive*, 2013. URL <https://github.com/nexr/RHive>. version 2.0. [p57]
- Pivotal Inc. *Greenplum Database*, 2013a. URL <http://www.gopivotal.com/products/pivotal-greenplum-database>. version 4.2.4. [p58]
- Pivotal Inc. *HAWQ*, 2013b. URL <http://www.gopivotal.com/pivotal-products/pivotal-data-fabric/pivotal-hd>. version 1.0. [p57]
- Pivotal Inc. *MADlib*, 2013c. URL <http://madlib.net>. version 1.4. [p57]
- Pivotal Inc. *PivotalR*, 2013d. URL <http://cran.r-project.org/web/packages/PivotalR/index.html>. version 0.1.8. [p57, 63]
- 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/>. [p]
- Revolution Analytics Company. *RHadoop*, 2012. URL <https://github.com/RevolutionAnalytics/RHadoop/wiki>. [p57]
- RStudio Inc. *shiny: Web Application Framework for R*, 2013. URL <http://CRAN.R-project.org/package=shiny>. R package version 0.6.0. [p58]
- Teradata Corporation. *teradataR*. Teradata Corporation, 2012. URL <http://downloads.teradata.com/download/applications/teradata-r>. version 1.0.1. [p57]
- The Apache Software Foundation. *HDFS*, 2013. URL http://hadoop.apache.org/docs/stable/hdfs_user_guide.html. version 0.23. [p57]
- H. Wickham and R. Francois. *dplyr: a grammar of data manipulation*, 2013. URL <https://github.com/hadley/dplyr>. version 0.1. [p57]

Hai Qian
Pivotal Inc.
3495 Deer Creek Rd., Palo Alto, CA 94304
USA hqian@gopivotal.com

rotations: An R Package for $SO(3)$ Data

by Bryan Stanfill, Heike Hofmann and Ulrike Genschel

Abstract In this article we introduce the **rotations** package which provides users with the ability to simulate, analyze and visualize three-dimensional rotation data. More specifically it includes four commonly used distributions from which to simulate data, four estimators of the central orientation, six confidence region estimation procedures and two approaches to visualizing rotation data. All of these features are available for two different parameterizations of rotations: three-by-three matrices and quaternions. In addition, two datasets are included that illustrate the use of rotation data in practice.

Introduction

Data in the form of three-dimensional rotations have applications in many scientific areas, such as biomedical engineering, computer vision, and geological and materials sciences where such data represent the positions of objects within a three-dimensional reference frame. For example, [Humbert et al. \(1996\)](#), [Bingham et al. \(2009\)](#) and [Bachmann et al. \(2010\)](#) apply rotation data to study the orientation of cubic crystals on the surfaces of metal. [Rancourt et al. \(2000\)](#) use rotations to represent variations in human movement while performing a task.

A common goal shared in the analysis of rotation data across all fields is to estimate the main or central orientation for a sample of rotations. More formally, let $SO(3)$ denote the rotation group, which consists of all real-valued 3×3 matrices \mathbf{R} with determinant equal to +1. Then observations $\mathbf{R}_1, \dots, \mathbf{R}_n \in SO(3)$ can be conceptualized as a random sample from a *location model*

$$\mathbf{R}_i = \mathbf{S} \mathbf{E}_i, \quad i = 1, \dots, n, \quad (1)$$

where $\mathbf{S} \in SO(3)$ is the *fixed* parameter of interest indicating the central orientation, and $\mathbf{E}_1, \dots, \mathbf{E}_n \in SO(3)$ denote i.i.d. *random* rotations which symmetrically perturb \mathbf{S} . Model (1) is a rotation-matrix analog of a location model for scalar data $Y_i = \mu + e_i$, where $\mu \in \mathbb{R}$ denotes a mean and $e_i \in \mathbb{R}$ denotes an additive error symmetrically distributed around zero.

Assuming the perturbations \mathbf{E}_i symmetrically perturb \mathbf{S} implies that the observations \mathbf{R}_i have no preferred direction relative to \mathbf{S} and that $E(\mathbf{R}_i) = c\mathbf{S}$ for some $c \in \mathbb{R}^+$ for all i . Also note that under the symmetry assumption, (1) could be equivalently specified as $\mathbf{R}_i = \mathbf{E}_i \mathbf{S}$, though the form given in (1) is the most common form in the literature (see [Bingham et al. 2009](#) for details).

While there is a multitude of packages and functions available in R to estimate the mean in a location model, the toolbox for rotational data is limited. The **orientlib** ([Murdoch, 2003](#)) package includes the definition of an orientation class along with a few methods to summarize and visualize rotation data. A strength of the **orientlib** package is its thorough exploration of rotation representations, but the estimation and visualization techniques are lacking and no methods for inference are available. The **onion** ([Hankin, 2011](#)) package includes functions for rotation algebra but only the quaternion form is available and data analysis is not possible. The **uarsbayes** ([Qiu, 2013](#)) package includes functions for data generation and Bayes inference but this package is currently not publicly available. Packages for circular and spherical data, e.g. **circular** ([Agostinelli and Lund, 2013](#)) and **SpherWave** ([Oh and Kim, 2013](#)), can possibly be used but their extension to rotation data is not straightforward.

The **rotations** ([Stanfill et al., 2014a](#)) package fills this void by providing users with the tools necessary to simulate rotations from (1) with four distribution choices for the perturbation matrices \mathbf{E}_i . Estimation and inference for \mathbf{S} in (1) is available along with two visualization techniques. The remainder of this manuscript introduces rotation data more fully and discusses the ways they are handled by the **rotations** package. For the latest on this package as well as a full list of available functions, see `help(package = "rotations")`.

Rotation parameterizations

Several parameterizations of rotations exist. We consider two of the most commonly used: orthogonal 3×3 matrices with determinant one and four-dimensional unit vectors called *quaternions*. The **rotations** package allows for both parameterizations as input as well as transforming one into the other. We will briefly discuss each:

Matrix form

Rotations in three-dimensions can be represented by 3×3 orthogonal matrices with determinant one. Matrices with these characteristics form a group called the *special orthogonal group*, or *rotation group*, denoted $SO(3)$. Every element in $SO(3)$ is associated with a skew-symmetric matrix $\Phi(W)$ where

$$\Phi(W) = \begin{pmatrix} 0 & -w_3 & w_2 \\ w_3 & 0 & -w_1 \\ -w_2 & w_1 & 0 \end{pmatrix}$$

and $W \in \mathbb{R}^3$. Applying the exponential operator to the matrix $\Phi(W)$ results in the rotation R

$$R = \exp[\Phi(W)] = \sum_{k=0}^{\infty} \frac{[\Phi(W)]^k}{k!}. \quad (2)$$

Since $\Phi(W)$ is skew-symmetric, it can be shown that (2) reduces to

$$R = \cos(r)I_{3 \times 3} + \sin(r)\Phi(U) + [1 - \cos(r)]UU^T, \quad (3)$$

where $r = \|W\|$, $U = W/\|W\|$. In the material sciences literature r and $U \in \mathbb{R}^3$ are termed the *misorientation angle* and *misorientation axis*, respectively.

Given a rotation matrix R one can find the associated skew-symmetric matrix $\Phi(W)$ by applying the logarithm operator defined by

$$\text{Log}(R) = \begin{cases} 0 & \text{if } \theta = 0 \\ \frac{r}{2\sin r} (R - R^T) & \text{otherwise,} \end{cases} \quad (4)$$

where $r \in [-\pi, \pi)$ satisfies $\text{tr}(R) = 1 + 2\cos r$ and $\text{tr}(\cdot)$ denotes the trace of a matrix. For more on the correspondence between $SO(3)$ and skew-symmetric matrices see Stanfill et al. (2013).

The **rotations** package defines the S3 class "S03", which internally stores a sample of n rotations as a $n \times 9$ matrix. If $n = 1$ then an object of class "S03" is printed as a 3×3 matrix but for $n > 1$ the $n \times 9$ matrix is printed. Objects can be coerced into, or tested for the class "S03" with the `as.S03` and `is.S03` functions, respectively. Any object passed to `is.S03` is tested for three characteristics: dimensionality, orthogonality and determinant one.

The `as.S03` function coerces the input into the class "S03". There are three types of input supported by the `as.S03` function. Given a single angle r and axis U , `as.S03` will form a rotation matrix according to (3). Equivalently one could supply a three-dimensional vector W , then the length of that vector will be taken to be the angle of rotation $r = \|W\|$ and the axis is taken to be the unit-vector in the direction of W , i.e. $U = W/\|W\|$. One can also supply a rotation Q in the quaternion representation. The `as.S03` function will return the matrix equivalent of Q . For all input types the function `as.S03` returns an $n \times 9$ matrix of class "S03" where each row corresponds to a rotation matrix. Below we illustrate the use of the `as.S03` function by constructing the 3×3 matrix associated with a 90° rotation about the y -axis, i.e. $r = \pi/2$ and $U = (0, 1, 0)$. In this example and all that follow, we have rounded the output to three digits for compactness.

```
> r <- pi/2
> U <- c(0, 1, 0)
> W <- U * r
> R <- as.S03(W)
> R

      [,1] [,2] [,3]
[1,]    0    0    1
[2,]    0    1    0
[3,]   -1    0    0

> identical(R, as.S03(U, r))

[1] TRUE
```

Given a rotation matrix R , the functions `mis.angle` and `mis.axis` will determine the misorientation angle and axis of an object with class "S03" as illustrated in the next example.

```
> mis.angle(R) * 2/pi
```

```
[1] 1
> mis.axis(R)
      [,1] [,2] [,3]
[1,]    0    1    0
```

Quaternion form

Quaternions are unit vectors in \mathbb{R}^4 that are commonly written as

$$\mathbf{Q} = x_1 + x_2i + x_3j + x_4k, \quad (5)$$

where $x_l \in [-1, 1]$ for $l = 1, 2, 3, 4$ and $i^2 = j^2 = k^2 = ijk = -1$. We can write $\mathbf{Q} = (s, \mathbf{V})$ as tuple of the scalar s for coefficient 1 and vector \mathbf{V} for the remaining coefficients, i.e. $s = x_1$ and $\mathbf{V} = (x_2, x_3, x_4)$.

A rotation around axis \mathbf{U} by angle r translates to $\mathbf{Q} = (s, \mathbf{V})$ with

$$s = \cos(r/2), \quad \mathbf{V} = \mathbf{U} \sin(r/2).$$

Note that rotations in quaternion form are over-parametrized: \mathbf{Q} and $-\mathbf{Q}$ represent equivalent rotations. This ambiguity has no impact on the distributional models, parameter estimation or inference methods to follow. Hence, for consistency, the **rotations** package only generates quaternions satisfying $x_1 \geq 0$. Data provided by the user does not need to satisfy this condition however.

The S3 class "Q4" is defined for the quaternion representation of rotations. All the functionality of the "S03" class also exists for the "Q4" class, e.g. `is.Q4` and `as.Q4` will test for and coerce to class "Q4", respectively. Internally, a sample of n quaternions is stored in the form of a $n \times 4$ matrix with each row a unit vector. Single quaternions are printed according to the representation in (5) (see example below) while a sample of size n is printed as a $n \times 4$ matrix with column names Real, i, j and k to distinguish between the four components.

The following code creates the same rotation from the previous section in the form of a quaternion with the `as.Q4` function. This function works much the same way as the `as.S03` function in terms of possible inputs but returns a vector of length four of the class "Q4".

```
> as.Q4(U, r)
0.707 + 0 * i + 0.707 * j + 0 * k
> as.Q4(as.S03(U, r))
0.707 + 0 * i + 0.707 * j + 0 * k
```

Data generation

If the rotation $E_i \in SO(3)$ from (1) has an axis \mathbf{U} that is uniformly distributed on the unit sphere and an angle r that is independently distributed about zero according to some symmetric distribution function then E_i is said to belong to the *uniform-axis random spin*, or *UARS*, class of distributions. From Bingham et al. (2009) the density for E_i is given by

$$f(E_i|\kappa) = \frac{4\pi}{3 - \text{tr}(E_i)} C \left(\arccos \left\{ \frac{\text{tr}(E_i) - 1}{2} \right\} \middle| \kappa \right), \quad (6)$$

where $C(\cdot|\kappa)$ is the distribution function associated with the angle of rotation r with concentration parameter κ . Members of the UARS family of distributions are differentiated based on the angular distribution $C(\cdot|\kappa)$.

The **rotations** package gives the user access to four members of the UARS class. Each member is differentiated by the distribution function for r : the uniform, the matrix Fisher (Langevin, 1905; Downs, 1972; Khatri and Mardia, 1977; Jupp and Mardia, 1979), the Cayley (Schaeben, 1997; León et al., 2006) and the circular-von Mises distribution (Bingham et al., 2009). Note: probability distribution functions on $SO(3)$ such as (6) are defined with respect to the Haar measure, which we denote by λ . That is, the expectation of a random rotation $\mathbf{R} \in SO(3)$ with corresponding misorientation angle r is given by $E(\mathbf{R}) = \int_{\Omega} \mathbf{R} f(\mathbf{R}|\kappa) d\lambda$ where $\Omega = SO(3)$, $d\lambda = [1 - \cos(r)]dr/(2\pi)$ and dr is the derivative of r with respect to the Lebesgue measure. Because the Haar measure acts as the uniform measure on $SO(3)$ and $\lambda(\Omega) = 1$, then the angular distribution $C(r) = [1 - \cos(r)]/(2\pi)$ is referred to as the

uniform distribution for misorientation angles r and has been included in the **rotations** package under the name `.haar` (see Table 1).

The spread of the Cayley, matrix Fisher and circular-von Mises distributions is controlled by the concentration parameter κ . Concentration is a distribution specific quantity and is not comparable across different distributions. To make comparisons across distributions possible we also allow for specification of the circular variance, which is defined as $\nu = 1 - E[\cos(r)]$ where $E[\cos(r)]$ is often referred to as the *mean resultant length* (Fisher, 1996). The form of each angular distribution along with the circular variance as a function of the concentration parameter is given in Table 1.

Name	Density $C(r \kappa)$	Circular variance ν	Function
Uniform	$\frac{1-\cos(r)}{2\pi}$	$\frac{3}{2}$	<code>.haar</code>
Cayley	$\frac{\Gamma(\kappa+2)(1+\cos r)^\kappa(1-\cos r)}{2^{(\kappa+1)}\sqrt{\pi}\Gamma(\kappa+1/2)}$	$\frac{3}{\kappa+2}$	<code>.cayley</code>
matrix Fisher	$\frac{[1-\cos(r)] \exp[2\kappa \cos(r)]}{2\pi[I_0(2\kappa)-I_1(2\kappa)]}$	$\frac{3I_0(2\kappa)-4I_1(2\kappa)+I_2(2\kappa)}{2[I_0(2\kappa)-I_1(2\kappa)]}$	<code>.fisher</code>
circular-von Mises	$\frac{\exp[\kappa \cos(r)]}{2\pi I_0(\kappa)}$	$\frac{I_0(\kappa)-I_1(\kappa)}{I_0(\kappa)}$	<code>.vmises</code>

Table 1: Circular densities and circular variance ν ; $I_i(\cdot)$ represents the modified Bessel function of order i and $\Gamma(\cdot)$ is the gamma function.

For a given concentration d , p and r take the same meaning as for the more familiar distributions such as `dnorm`. To simulate a sample of $SO(3)$ data, the `ruars` function takes arguments `n`, `rangle`, and `kappa` to specify the sample size, angular distribution and concentration as shown below. Alternatively, one can specify the circular variance ν . Circular variance is used in the event that both circular variance and concentration are provided. The `space` argument determines the parameterization to form. When a sample of rotations is printed then a $n \times 9$ matrix is printed with column titles that specify which element of the matrix each column corresponds to. For example, the $R_{\{1,1\}}$ element of a rotation matrix is printed under the column heading `R11` as illustrated below.

```
> Rs <- ruars(n = 20, rangle = rcayley, kappa = 1, space = "S03")
> Qs <- ruars(n = 20, rangle = rcayley, kappa = 1, space = "Q4")
> Rs <- ruars(n = 20, rangle = rcayley, nu = 1, space = "S03")
> Qs <- ruars(n = 20, rangle = rcayley, nu = 1, space = "Q4")
> head(Rs, 3)
```

```
      R11  R21  R31  R12  R22  R32  R13  R23  R33
[1,] -0.425 -0.850 0.310 0.475 -0.501 -0.723 0.770 -0.160 0.617
[2,] -0.564 -0.733 0.379 0.745 -0.256 0.615 -0.354 0.630 0.691
[3,] 0.087 -0.716 0.692 0.117 0.698 0.707 -0.989 0.019 0.145
```

Data analysis

In this section we present functions in the **rotations** package to compute point estimates and confidence regions for the central orientation S .

Estimation of central orientation

Given a sample of n observations R_1, \dots, R_n generated according to (1), the **rotations** package offers four built-in ways to estimate the central orientation S . These estimators are either Riemannian- or Euclidean-based in geometry and use either the L_1 - or L_2 - norm, i.e. they are median- or mean-type. We briefly discuss how the choice of geometry affects estimation of S .

The choice of geometry results in two different metrics to measure the distance between rotation matrices R_1 and $R_2 \in SO(3)$. The Euclidean distance, d_E , between two rotations is defined by

$$d_E(R_1, R_2) = \|R_1 - R_2\|_F,$$

where $\|A\|_F = \sqrt{\text{tr}(A^T A)}$ denotes the Frobenius norm. The Euclidean distance between two rotation matrices corresponds to the length of the shortest path in $\mathbb{R}^{3 \times 3}$ that connects them and is therefore an

extrinsic distance metric.

Estimators based on the Euclidean distance form the class of *projected* estimators. The name is derived from the method used to compute these estimators. That is, each estimator in this class is the projection of the the generic 3×3 matrix that minimizes the loss function into $SO(3)$. For an object with class "S03" the median or mean function with argument type = "projected" will return a 3×3 matrix in $SO(3)$ that minimizes the first- or second-order loss function, respectively.

By staying in the Riemannian space $SO(3)$ the natural distance metric becomes the Riemannian (or geodesic) distance, d_R , which for two rotations $R_1, R_2 \in SO(3)$ is defined as

$$d_R(R_1, R_2) = \frac{1}{\sqrt{2}} \left\| \text{Log} \left(R_1^\top R_2 \right) \right\|_F = |r|,$$

where $\text{Log}(R)$ denotes the logarithm of R defined in (4) and $r \in [-\pi, \pi)$ is the misorientation angle of $R_1^\top R_2$. The Riemannian distance corresponds to the length of the shortest path that connects R_1 and R_2 within the space $SO(3)$ and is therefore an *intrinsic* distance metric. For this reason, the Riemannian distance is often considered the more natural metric on $SO(3)$. As demonstrated in Stanfill et al. (2013), the Euclidean and Riemannian distances are related by $d_E(R_1, R_2) = 2\sqrt{2} \sin [d_R(R_1, R_2)/2]$.

Estimators based on the Riemannian distance metric are called *geometric* estimators because they preserve the geometry of $SO(3)$. These can be computed using the mean and median functions with the argument type = "geometric". Table 2 summarizes the four estimators including their formal definition and how they can be computed.

The estimators in Table 2 find estimates based on minimization of L_1 - and L_2 -norms in the chosen geometry. The function `gradient.search` provides the option to optimize for any other arbitrary minimization criterion. As the name suggests, the minimization is done along the gradient of the minimization function in the rotation space. Starting from an initial, user-specified rotation, the algorithm finds a (local) minimum by stepping iteratively in the direction of the steepest descent. Step size is regulated internally by adjusting for curvature of the minimization function.

We highlight this process in the example below. The function `L1.error` is defined to minimize the intrinsic L_1 -norm, the result from the optimization should therefore agree with the geometric median of the sample. In fact, the difference between the two results is at the same level as the minimal difference (`minerr`) used for convergence of the gradient search. What is gained in flexibility of the optimization is, of course, paid for in terms of speed: the built-in median function is faster by far than the gradient search.

Also illustrated in the example below is the `rot.dist` function, which computes the distance between two objects of class "S03", e.g. `R1` and `R2`. The argument `method` specifies which type of distance to compute: the "extrinsic" option will return the Euclidean distance and the "intrinsic" option will return the Riemannian distance. If `R1` is an $n \times 9$ matrix representing a sample of rotations, then `rot.dist` will return a vector of length n where the i th element represents the specified distance between `R2` and the i th row of `R1`.

```
> # error function definition
> L1.error <- function(sample, Shat) {
+   sum(rot.dist(sample, Shat, method = "intrinsic", p = 1))
+ }
> cayley.sample <- ruars(n = 50, rangle = rcayley, nu = 1, space = "S03")
> # gradient based optimization
> system.time(SL1 <- gradient.search(cayley.sample, L1.error))

   user  system elapsed 
 3.464   0.007   3.473 

> # in-built function
> system.time(S <- median(cayley.sample, type = "geometric"))

   user  system elapsed 
0.004   0.000   0.005 

> rot.dist(S, SL1$Shat)

[1] 1.492e-05
```

Estimator name	Definition	Code
Projected Mean	$\hat{S}_E = \operatorname{argmin}_{S \in SO(3)} \sum_{i=1}^n d_E^2(S, R_i)$	<code>mean(Rs, type = "projected")</code>
Projected Median	$\tilde{S}_E = \operatorname{argmin}_{S \in SO(3)} \sum_{i=1}^n d_E(S, R_i)$	<code>median(Rs, type = "projected")</code>
Geometric Mean	$\hat{S}_R = \operatorname{argmin}_{S \in SO(3)} \sum_{i=1}^n d_R^2(S, R_i)$	<code>mean(Rs, type = "geometric")</code>
Geometric Median	$\tilde{S}_R = \operatorname{argmin}_{S \in SO(3)} \sum_{i=1}^n d_R(S, R_i)$	<code>median(Rs, type = "geometric")</code>

Table 2: A summary of the estimators included in the **rotations** package. Rs is a sample of n rotations with class "S03" or "Q4".

Confidence regions

Asymptotic results for the distribution of the projected mean \hat{S}_E and median \tilde{S}_E can be used to construct confidence regions for the central orientation S . In the literature two approaches are available to justify the limiting distribution of the vector in \mathbb{R}^3 associated with the centered estimator through (2). More specifically, the vector $\sqrt{n}\hat{h}$ has been shown to have a trivariate normal distribution where $\hat{h} \in \mathbb{R}^3$ satisfies

$$\exp \left[\Phi \left(\hat{h} \right) \right] = S^\top \hat{S}_E.$$

The first approach transforms a result from directional statistics while the second uses M -estimation theory in $SO(3)$ directly. A summary of these methods is given next.

In the context of directional statistics, [Prentice \(1984\)](#) used results found in [Tyler \(1981\)](#) and the fact that \hat{S}_E is a function of the spectral decomposition of $\bar{R} = \sum_{i=1}^n R_i / n$ in order to justify a multivariate normal limiting distribution for the scaled vector $\sqrt{n}\hat{h}$. Unsatisfied with the coverage rate achieved by [Prentice \(1986\)](#), [Fisher et al. \(1996\)](#) proposed a pivotal bootstrap procedure that results in coverage rates closer to the nominal level for small samples. A transformation from unit vectors in \mathbb{R}^d to rotation matrices is required in order to apply the results of [Prentice \(1984\)](#) and [Fisher et al. \(1996\)](#) to $SO(3)$, therefore they are called *transformation-based*. The projected median \tilde{S}_E cannot be expressed as a function of the sample spectral decomposition, therefore this approach cannot be used to create confidence regions based on \tilde{S}_E .

It has also been shown that both estimators \hat{S}_E and \tilde{S}_E are M -estimators, which motivates a direct approach to confidence region estimation in $SO(3)$ ([Chang and Rivest, 2001](#)). In [Stanfill et al. \(2014b\)](#), a pivotal bootstrap method based on the direct approach was implemented to improve coverage rates in small samples. Because the results in [Chang and Rivest \(2001\)](#) and [Stanfill et al. \(2014b\)](#) deal with $SO(3)$ data directly, this approach is called *direct*.

The six possible confidence regions that result from these two methods are available through the wrapper function `region`. They are differentiated based on the method, type and estimator arguments. Set `estimator = "mean"` or `estimator = "median"` to estimate a region based on \hat{S}_E or \tilde{S}_E , respectively. For \hat{S}_E one can choose `method = "transformation"` for the transformation-based methods or `method = "direct"` for the direct method. Since the transformation-based methods cannot be applied to \tilde{S}_E an error is returned if `estimator = "median"` and `method = "transformation"` are combined. A bootstrap version of the specified method is implemented if `type = "bootstrap"` or the normal limiting distribution can be chosen with `type = "asymptotic"`. If a bootstrap type region is specified one can additionally specify the bootstrap sample size with the `m` argument, which is set to 300 by default. Regardless of the method and type chosen a single value is returned on the interval $(0, \pi]$. This value corresponds to the radius of the confidence region centered at each of the axes of the specified estimator.

In the example code below a sample of $n = 50$ rotations are drawn from the Cayley-UARS($I_{3 \times 3}, \kappa = 10$) distribution then the four types of confidence regions based on the direct approach are constructed. For a graphical representation of this dataset along with an interpretation of the confidence regions see [Figure 1b](#).

```
> Rs <- ruars(50, rcayley, kappa = 10)
```

```

> region(Rs, method = "direct", type = "asymptotic",
+       estimator = "mean", alp = 0.05)

[1] 0.189

> region(Rs, method = "direct", type = "bootstrap", estimator = "mean",
+       alp = 0.05, m = 300)

[1] 0.201

> region(Rs, method = "direct", type = "asymptotic",
+       estimator = "median", alp = 0.05)

[1] 0.201

> region(Rs, method = "direct", type = "bootstrap", estimator = "median",
+       alp = 0.05, m = 300)

[1] 0.249

```

Visualizations

The **rotations** package offers two methods to visualize rotation data in three-dimensions. Because rotation matrices are orthogonal, each column of a rotation matrix has length one and is perpendicular to the other axes. Therefore each column of a rotation matrix can be illustrated as a point on the surface of a unit sphere, which represents the position of the x -, y - or z -axis for that rotation matrix. Since each sphere represents one of the three axes, three spheres are required to fully visualize a sample of rotations. Though the use of separate spheres to represent each axis can be seen as a disadvantage, the proposed visualization method makes the idea of a central orientation and a confidence region interpretable.

An existing function that can be used to illustrate rotation data is the `boat3d` function included in the **orientlib** package. Given a sample of rotations, the `boat3d` function produces either a static or interactive three-dimensional boat to represent the provided data. If only one rotation is of interest, the `boat3d` function is superior to the proposed method because it conveniently illustrates rotational data in a single image. If multiple rotations are provided, however, the `boat3d` function will produce separate side-by-side boats, which can be hard to interpret. In addition, the illustration of a estimated central orientation or a confidence region in $SO(3)$ with the `boat3d` function is not presently possible.

The **rotations** package can be used to produce high-quality static plots within the framework of the **ggplot2** package (Wickham, 2009). Static plots are specifically designed for datasets that are highly concentrated and for use in presentations or publications. Alternatively, the **rotations** package can produce interactive plots using functions included in the **sphereplot** package (Robotham, 2013). Interactive plots are designed so that the user can explore a dataset and visualize a diffuse sample.

Calling the `plot` function with a "S03" or "Q4" object will result in an interactive or static sphere, differentiated by setting the argument `interactive` to `TRUE` or `FALSE`, respectively. The `center` argument defines the center of the plot and is usually set to the identity rotation `id.S03` or an estimate of the central orientation, e.g. `mean(Rs)`. The user can specify which columns to visualize with the `col` argument with options 1, 2 and 3 representing the x -, y - and z - axes, respectively. For static plots, multiple axes can be displayed simultaneously by supplying a vector to `col`; only one column will be displayed at a time for interactive plots. Also available to static plots is the argument `to_range`, which when set to `TRUE` will display the portion of the sphere where the observations are present.

All four estimates of the central orientation can be plotted along with a sample of rotations. Setting the argument `estimates_show = "all"` will display all four simultaneously. If only a few estimates are of interest then any combination of "proj.mean", "proj.median", "geom.mean" or "geom.median" are valid inputs. The estimators are indicated by color and a legend is provided, see Figure 1a. Finally, the `mean_regions` and `median_regions` options allow the user to draw a circle on the surface of the sphere representing the confidence region for that axis, centered at \hat{S}_E and \tilde{S}_E respectively. If estimators are plotted along with the different regions in static plots then shapes represent the estimators and colors represent the region methods, see Figure 1b, while regions and estimators are always distinguished by colors for the interactive plots. Given the sample of rotations generated in a previous example, the example below illustrates how to produce static plots using the `plot` function for objects of class "S03" and Figure 1 illustrates the results of these commands.


```
> plot(Rs, center = mean(Rs), col = 1, show_estimates = "all",
+       interactive = FALSE)
> plot(Rs, center = mean(Rs), col = 1, show_estimates = "proj.mean",
+       mean_regions = "all", alp = .05, interactive = FALSE)
```

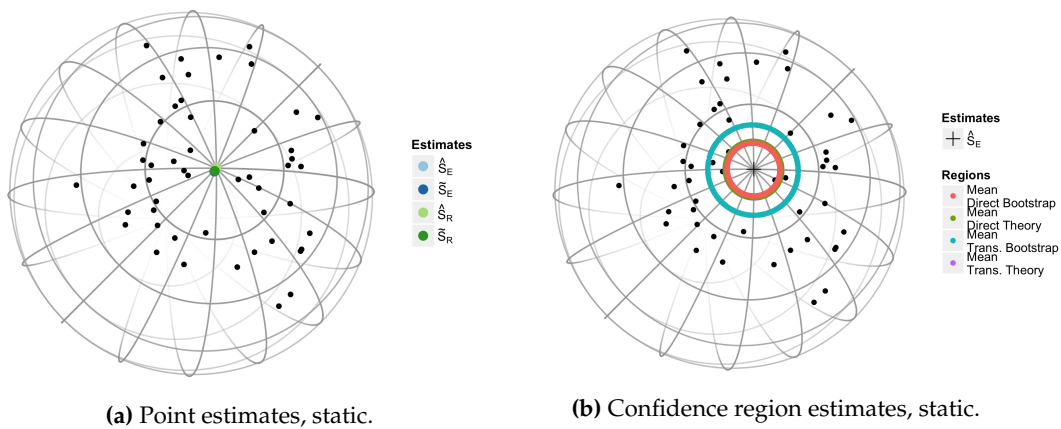


Figure 1: The x -axis of a random sample from the Cayley-UARS distribution with $\kappa = 1$, $n = 50$. All for point estimates are displayed in (a) and all three region methods along with the projected mean are in (b).

Datasets

Datasets `drill` and `nickel` are included in the **rotations** package to illustrate how the two representations of orientation data discussed here are used in practice. The `drill` dataset was collected to assess variation in human movement while performing a task (Rancourt, 1995). Eight subjects drilled into a metal plate while being monitored by infrared cameras. Quaternions are used to represent the orientation of each subjects' wrist, elbow and shoulder in one of six positions. For some subjects several replicates are available. See Rancourt et al. (2000) for one approach to analyzing these data. In the example below we load the `drill` dataset, coerce the observations for subject one's wrist into a form usable by the **rotations** package via `as.Q4`, then estimate the central orientation with the projected mean.

```
> data(drill)
> head(drill)
```

	Subject	Joint	Position	Replicate	Q1	Q2	Q3	Q4
1	1	Wrist	1	1	0.944	-0.192	-0.156	0.217
2	1	Wrist	1	2	0.974	-0.120	-0.111	0.158
3	1	Wrist	1	3	0.965	-0.133	-0.141	0.177
4	1	Wrist	1	4	0.956	-0.134	-0.115	0.233
5	1	Wrist	1	5	0.953	-0.199	-0.061	0.222
6	1	Wrist	2	1	0.963	-0.159	-0.127	0.177

```
> Subj1Wrist<-subset(drill, Subject == '1' & Joint == 'Wrist')
> Subj1Wdata <- as.Q4(Subj1Wrist[, 5:8])
> mean(Subj1Wdata)
```

$$0.987 - 0.070 * i - 0.134 * j + 0.049 * k$$

In the `nickel` dataset, rotation matrices are used to represent the orientation of cubic crystals on the surface of a nickel sample measured with Electron Backscatter Diffraction. Each location on the surface of the nickel is identified by the `xpos` and `ypos` columns while the `rep` column identifies which of the fourteen replicate scans that measurement corresponds to. The last nine columns, denoted `v1-v9`, represent the elements of the rotation matrix at that location in vector form. See Bingham et al. (2009, 2010) and Stanfill et al. (2013) for more details. In the example below we estimate the central orientation at location one.

```
> data(nickel)
> head(nickel[, 1:6])
```



```

      xpos ypos location rep      V1      V2
1      0 0.346          1  1 -0.648 0.686
2      0 0.346          1  2 -0.645 0.688
3      0 0.346          1  3 -0.645 0.688
4      0 0.346          1  4 -0.646 0.688
5      0 0.346          1  5 -0.646 0.686
6      0 0.346          1  6 -0.644 0.690

> Location1<-subset(nickel, location == 1)
> Loc1data<-as.S03(Location1[, 5:13])
> mean(Loc1data)

      [,1] [,2] [,3]
[1,] -0.645 -0.286 -0.708
[2,]  0.687 -0.623 -0.374
[3,] -0.334 -0.728  0.599

```

Summary

In this manuscript we introduced the **rotations** package and demonstrated how it can be used to generate, analyze and visualize rotation data. The **rotations** package is compatible with the quaternion specific **onion** package by applying its `as.quaternion` function to a transposed "Q4" object. Connecting to the **onion** package gives the user access to a wide range of algebraic functions unique to quaternions. Also compatible with the **rotations** package is the **orientlib** package, which includes additional parameterizations of rotations. To translate rotation matrices generated by the **rotations** package into a form usable by the **orientlib** package, first coerce a "S03" object into a matrix of the same dimension, i.e. $n \times 9$, then apply the `rotvector` function provided by the **orientlib** package. Quaternions are defined in the **orientlib** package by $Q = x_1i + x_2j + x_3k + x_4$, cf. (5), which may lead to confusion when translating quaternions between the **orientlib** package and either of the **onion** or **rotations** packages. Below is a demonstration of how quaternions and rotation matrices generated by the **rotations** package can be translated into a form usable by the **onion** and **orientlib** packages, respectively. See `help(package = "onion")` and `help(package = "orientlib")` for more on these packages.

```

> Qs <- ruars(20, rcayley, space = 'Q4')
> Rs <- as.S03(Qs)
> suppressMessages(require(onion))
> onionQs <- as.quaternion(t(Qs))
> suppressMessages(require(orientlib))
> orientRs <- rotvector(matrix(Rs, ncol = 9))

```

Computational speed of the **rotations** package has been enhanced through use of the **Rcpp** and **RcppArmadillo** packages (Eddelbuettel, 2013; Eddelbuettel and Sanderson, 2014). In future versions of the package we plan to extend the parameterization and estimator sections to include robust estimators currently being developed by the authors.

Acknowledgements

We would like to thank the reviewers for their comments and suggestions. The **rotations** package and this article have benefited greatly from their time and effort.

Bibliography

- C. Agostinelli and U. Lund. *circular: Circular Statistics*, 2013. URL <http://CRAN.R-project.org/package=circular>. R package version 0.4-7. [p68]
- F. Bachmann, R. Hielscher, P. Jupp, W. Pantleon, H. Schaebe, and E. Wegert. Inferential statistics of electron backscatter diffraction data from within individual crystalline grains. *Journal of Applied Crystallography*, 43(6):1338–1355, 2010. [p68]
- M. A. Bingham, D. J. Nordman, and S. B. Vardeman. Modeling and inference for measured crystal orientations and a tractable class of symmetric distributions for rotations in three dimensions. *Journal of the American Statistical Association*, 104(488):1385–1397, 2009. [p68, 70, 75]

- M. A. Bingham, B. K. Lograsso, and F. C. Laabs. A statistical analysis of the variation in measured crystal orientations obtained through electron backscatter diffraction. *Ultramicroscopy*, 110(10):1312–1319, 2010. [p75]
- T. Chang and L.-P. Rivest. M-estimation for location and regression parameters in group models: A case study using Stiefel manifolds. *Annals of statistics*, 29(3):784–814, 2001. [p73]
- T. Downs. Orientation statistics. *Biometrika*, 59(3):665–676, 1972. [p70]
- D. Eddelbuettel. *Seamless R and C++ Integration with Rcpp*. Springer-Verlag, New York, 2013. ISBN 978-1-4614-6867-7. [p76]
- D. Eddelbuettel and C. Sanderson. Rcpparmadillo: Accelerating R with high-performance C++ linear algebra. *Computational Statistics and Data Analysis*, 71:1054–1063, Mar. 2014. URL <http://dx.doi.org/10.1016/j.csda.2013.02.005>. [p76]
- N. I. Fisher. *Statistical Analysis of Circular Data*. Cambridge University Press, 1996. ISBN 0521568900. [p71]
- N. I. Fisher, P. Hall, B.-Y. Jing, and A. T. Wood. Improved pivotal methods for constructing confidence regions with directional data. *Journal of the American Statistical Association*, 91(435):1062–1070, 1996. [p73]
- R. K. S. Hankin. *onion: octonions and quaternions*, 2011. URL <http://CRAN.R-project.org/package=onion>. R package version 1.2-4. [p68]
- M. Humbert, N. Gey, J. Muller, and C. Esling. Determination of a mean orientation from a cloud of orientations. Application to electron back-scattering pattern measurements. *Journal of Applied Crystallography*, 29(6):662–666, 1996. [p68]
- P. Jupp and K. Mardia. Maximum likelihood estimators for the matrix von Mises-Fisher and Bingham distributions. *The Annals of Statistics*, 7(3):599–606, 1979. [p70]
- C. Khatri and K. Mardia. The von Mises-Fisher matrix distribution in orientation statistics. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):95–106, 1977. [p70]
- P. Langevin. Magnetism and the theory of the electron. *Annales de Chimie et de Physique*, 5:70, 1905. [p70]
- C. León, J. Massé, and L. Rivest. A statistical model for random rotations. *Journal of Multivariate Analysis*, 97(2):412–430, 2006. [p70]
- D. Murdoch. orientlib: An R package for orientation data. *Journal of Statistical Software*, 8(19):1–11, 2003. [p68]
- H. Oh and D. Kim. *SpherWave: Spherical Wavelets and SW-based Spatially Adaptive Methods*, 2013. URL <http://CRAN.R-project.org/package=SpherWave>. R package version 1.2.2. [p68]
- M. Prentice. A distribution-free method of interval estimation for unsigned directional data. *Biometrika*, 71(1):147–154, 1984. [p73]
- M. Prentice. Orientation statistics without parametric assumptions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 48(2):214–222, 1986. [p73]
- Y. Qiu. *Isotropic Distributions for 3-Dimensional Rotations and One-sample Bayes Inference*. PhD thesis, Iowa State University, 2013. [p68]
- D. Rancourt. *Arm Posture and Hand Mechanical Impedance in the Control of a Hand-held Power Drill*. Dissertation, MIT, 1995. [p75]
- D. Rancourt, L.-P. Rivest, and J. Asselin. Using orientation statistics to investigate variations in human kinematics. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 49(1):81–94, 2000. [p68, 75]
- A. Robotham. *sphereplot: Spherical Plotting*, 2013. URL <http://CRAN.R-project.org/package=sphereplot>. R package version 1.5. [p74]
- H. Schaeben. A simple standard orientation density function: The hyperspherical de la Vallée Poussin kernel. *Physica Status Solidi (B)*, 200(2):367–376, 1997. [p70]

- B. Stanfill, U. Genschel, and H. Hofmann. Point estimation of the central orientation of random rotations. *Technometrics*, 55(4):524–535, 2013. [p69, 72, 75]
- B. Stanfill, H. Hofmann, and U. Genschel. *rotations: Tools for Working with Rotation Data*, 2014a. URL <http://CRAN.R-project.org/package=rotations>. R package version 1.2. [p68]
- B. Stanfill, D. Nordman, H. Hofmann, U. Genschel, and J. Zhang. Nonparametric confidence regions for the central orientation of random rotations. Unpublished manuscript, 2014b. [p73]
- D. E. Tyler. Asymptotic inference for eigenvectors. *The Annals of Statistics*, 9(4):725–736, 1981. [p73]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag, New York, 2009. ISBN 978-0-387-98140-6. URL <http://had.co.nz/ggplot2/book>. [p74]

Bryan Stanfill
Department of Statistics
Iowa State University
Ames, IA 50011
stanfill@iastate.edu

Heike Hofmann
Department of Statistics
Iowa State University
Ames, IA 50011
hofmann@mail.iastate.edu

Ulrike Genschel
Department of Statistics
Iowa State University
Ames, IA 50011
ulrike@mail.iastate.edu

ROSE: A Package for Binary Imbalanced Learning

by Nicola Lunardon, Giovanna Menardi, and Nicola Torelli

Abstract The **ROSE** package provides functions to deal with binary classification problems in the presence of imbalanced classes. Artificial balanced samples are generated according to a smoothed bootstrap approach and allow for aiding both the phases of estimation and accuracy evaluation of a binary classifier in the presence of a rare class. Functions that implement more traditional remedies for the class imbalance and different metrics to evaluate accuracy are also provided. These are estimated by holdout, bootstrap, or cross-validation methods.

Introduction

Imbalanced learning is the heading which denotes the problem of supervised classification when one of the classes is rare over the sample. As class imbalance situations are pervasive in a plurality of fields and applications, the issue has received considerable attention recently. Numerous works have focused on warning about the heavy implications of neglecting the imbalance of classes, as well as proposing suitable solutions to relieve the problem. Nonetheless, there is a general lack of procedures and software explicitly aimed at handling imbalanced data and which can be readily adopted also by non expert users. To the best of our knowledge, in the R environment, only a few functions are designed for imbalanced learning. It is worth mentioning package **DMwR** (Torgo, 2010), which provides a specific function (`smote`) to aid the estimation of a classifier in the presence of class imbalance, in addition to extensive tools for data mining problems (among others, functions to compute evaluation metrics as well as different accuracy estimators). In addition, package **caret** (Kuhn, 2014) contains general functions to select and validate regression and classification problems and specifically addresses the issue of class imbalance with some naive functions (`downSample` and `upSample`).

These reasons motivate the **ROSE** package (Lunardon et al., 2013), which is intended to provide both standard and more refined tools to enhance the task of binary classification in an imbalanced setting. The package is designed around ROSE (Random Over-Sampling Examples), a smoothed bootstrap-based technique which has been recently proposed by Menardi and Torelli (2014). ROSE helps to relieve the seriousness of the effects of an imbalanced distribution of classes by aiding both the phases of model estimation and model assessment.

This paper is organized as follows: after a brief introduction to the problem of class imbalance and to the statistical foundations at the basis of the ROSE method, we provide an overview of the functions included in the package and illustrate their use with a numerical example.

The class imbalance problem

Without attempting a full discussion, we summarize here the main statistical issues emerging in imbalanced learning. The outline focuses on those aspects that are relevant for a full comprehension of the routines implemented in the package. The interested reader is invited to refer to Menardi and Torelli (2014) and the references therein for a deeper discussion and technical details.

The presence of a strong imbalance in the distribution of the response variable may lead to heavy consequences in pursuing a classification task, in both phases of model estimation and accuracy evaluation. Disregarding the specificities of different models, what typically happens is that classification rules are overwhelmed by the prevalent class and the rare examples are ignored.

Most of the current research on imbalanced classification focuses on proposing solutions to improve the model estimation step. The most common remedy to the imbalance problem involves altering the class distribution to obtain a more balanced sample. Remedies based on balancing the class distribution include various techniques of data resampling, such as random oversampling (with replacement) of the rare class and random undersampling (without replacement) of the prevalent class. Under the same hat of these balancing methods, we can also include the ones designed to generate new artificial examples that are ‘similar’, in a certain sense, to the rare observations. Generation of new artificial data that have not been previously observed reduces the risk of overfitting and improves the ability of generalization compromised by oversampling methods, which are bound to produce ties in the sample. As will be clarified subsequently, the ROSE technique can be rightly considered as following this route.

When a classification task is performed, evaluating the accuracy of the classifier plays a role that

is at least as important as model estimation, because the extent to which a classification rule may be operationally applied to real-world problems, for labeling new unobserved examples, depends on our ability to measure classification accuracy.

In the accuracy evaluation step, the first problem one has to face concerns the choice of the accuracy metric, since the use of standard measures, such as the overall accuracy, may yield misleading results. The choice of the evaluation measure has to be addressed in terms of some class-independent quantities, such as precision, recall or the F measure. For the operational computation of these measures, one should set a suitable threshold for the probability of belonging to the positive class, above which an example is predicted to be positive. In standard classification problems, this threshold is usually set to 0.5, but the same choice is not obvious in imbalanced learning, as it is likely that no examples are labeled as positive. Moreover, moving a threshold to smaller values is equivalent to assume a higher misclassification cost for the rare class, which is usually the case. To avoid an arbitrary choice of the threshold, a ROC curve can be adopted to measure the accuracy, because it plots the true positive rate versus the false positive rate as the classification threshold varies.

Apart from the choice of an adequate performance metric, a more serious problem in imbalanced learning concerns the estimation method for the selected accuracy measure. To this aim, standard practices are the resubstitution method, where the available data are used for both training and assessing the classifier or, more frequently, the holdout method, which consists of estimating the classifier over a training sample of data and assessing its accuracy on a test sample. In the presence of a class imbalance, often, there are not sufficient examples from the rare class for both training and testing the classifier. Additionally, the scarcity of data leads to estimates of the accuracy measure which are affected by a high variance and are then regarded as unreliable. On the other hand, the resubstitution method is known to lead to overoptimistic evaluation of learner accuracy. Then, alternative estimators of the accuracy measure have to be considered, as pointed out in the next section.

The ROSE strategy to deal with class imbalance

ROSE (Menardi and Torelli, 2014) provides a unified framework to deal simultaneously with the two above-mentioned problems of model estimation and accuracy evaluation in imbalanced learning. It builds on the generation of new artificial examples from the classes, according to a smoothed bootstrap approach (see, e.g., Efron and Tibshirani, 1993).

Consider a training set \mathbf{T}_n , of size n , whose generic row is the pair (\mathbf{x}_i, y_i) , $i = 1, \dots, n$. The class labels y_i belong to the set $\{\mathcal{Y}_0, \mathcal{Y}_1\}$, and \mathbf{x}_i are some related attributes supposed to be realizations of a random vector \mathbf{x} defined on \mathbb{R}^d , with an unknown probability density function $f(\mathbf{x})$. Let the number of units in class \mathcal{Y}_j , $j = 0, 1$, be denoted by $n_j < n$. The ROSE procedure for generating one new artificial example consists of the following steps:

1. Select $y^* = \mathcal{Y}_j$ with probability π_j .
2. Select $(\mathbf{x}_i, y_i) \in \mathbf{T}_n$, such that $y_i = y^*$, with probability $\frac{1}{n_j}$.
3. Sample \mathbf{x}^* from $K_{\mathbf{H}_j}(\cdot, \mathbf{x}_i)$, with $K_{\mathbf{H}_j}$ a probability distribution centered at \mathbf{x}_i and covariance matrix \mathbf{H}_j .

Essentially, we draw from the training set an observation belonging to one of the two classes, and generate a new example (\mathbf{x}^*, y^*) in its neighborhood, where the shape of the neighborhood is determined by the shape of the contour sets of K and its width is governed by \mathbf{H}_j .

It can be easily shown that, given selection of the class label \mathcal{Y}_j , the generation of new examples from \mathcal{Y}_j , according to ROSE, corresponds to the generation of data from the kernel density estimate of $f(\mathbf{x}|\mathcal{Y}_j)$, with kernel K and smoothing matrix \mathbf{H}_j (Menardi and Torelli, 2014). The choices of K and \mathbf{H}_j may be then addressed by the large specialized literature on kernel density estimation (see, e.g. Bowman and Azzalini, 1997). It is worthwhile to note that, for $\mathbf{H}_j \rightarrow 0$, ROSE collapses to a standard combination of over- and under-sampling.

Repeating steps 1 to 3 m times produces a new synthetic training set \mathbf{T}_m^* , of size m , where the imbalance level is defined by the probabilities π_j (if $\pi_j = 1/2$, then approximately the same number of examples belong to the two classes). The size m may be set to the original training set size n or chosen in any way.

Apart from enhancing the process of learning, the synthetic generation of new examples from an estimate of the conditional densities of the two classes may also aid the estimation of learner accuracy and overcome the limits of both resubstitution and the holdout method. Operationally, the use of ROSE for estimating learner accuracy may follow different schemes, which resemble standard accuracy estimators but claim a certain degree of originality. For example, a holdout version of ROSE would involve testing the classifier on the originally observed data after training it on the artificial training set \mathbf{T}_m^* . Alternatively, bootstrap or cross-validated versions of ROSE may be chosen as estimation

Table 1: Pseudo-code of the alternative uses of ROSE for model assessment.

Cross validation (leave-K-out) ROSE	Bootstrap ROSE
Split T_n into $Q=n/K$ sets $T_{K'}^1, \dots, T_K^Q$ for (i: 1 to Q) do get a ROSE sample T_m^{*i} from $T_n \setminus T_K^i$ estimate a classifier on T_m^{*i} make a prediction P_K^i on T_K^i end for compute accuracy of $\{P_K^1, \dots, P_K^Q\}$	for (b: 1 to B) do get a ROSE sample T_m^{*b} from T_n estimate a classifier on T_m^{*b} make a prediction P_n^b on T_n compute accuracy of P_n^b end for get the bootstrap distribution of the accuracy measure

methods, as illustrated in Table 1. An extensive simulation study in [Menardi and Torelli \(2014\)](#) has, in fact, shown that the holdout and bootstrap versions of ROSE tend to overestimate the accuracy, but their mean square error is lower than the one obtained by utilizing a standard holdout method. Hence, overall, these estimates are preferable.

Package illustration

Overview

The package provides a complete toolkit to tackle the problem of binary classification in the presence of imbalanced data. Functions are supplied to encompass all phases of the learning process: from model estimation to assessment of the accuracy of the classification. In the former phase, the user has to choose both the remedy to adopt for the class imbalance, and the classifier to estimate for the learning process. For the first aim, functions `ROSE` or `ovun.sample` can be adopted to balance the sample. One is allowed to choose among all the functions already implemented in R to build the desired binary classifier, such as `glm`, `rpart`, `nnet`, as well as user defined functions. Once a classifier has been trained, its accuracy has to be evaluated, which requires the choice of both an appropriate accuracy measure, and an estimation method that can provide a reliable estimate of such measure. Functions `roc.curve` and `accuracy.meas` implement the most commonly adopted measures of accuracy in imbalance learning, while function `ROSE.eval` provides a ROSE version of holdout, bootstrap or cross-validation estimates of the accuracy measures, as described above.

A summary of the functions provided by the package, classified according to the main tasks they are designed for, is listed in Table 2.

ROSE also includes the simulated data `hacide`, which are adopted here to illustrate the package. The workspace `hacide` consists of a bidimensional training set `hacide.train` and a test set `hacide.test`, amounting to 1000 and 250 rows, respectively. The binary label class (denoted as `cls`) has a heavily imbalanced distribution, with the positive examples occurring in approximately 2% of the cases. The rare class may be described as a depleted noisy semi-circle filled with the prevalent class, which is normally distributed and has elliptical contours. See the top-left panel of Figure 1 for an illustration.

Ignoring the imbalance

After loading the package and the data, we explore the training set structure:

```
> library(ROSE)
Loaded ROSE 0.0-3
```

Table 2: Summary of the functions in package **ROSE**

Data balancing	<code>ROSE</code>	<code>ovun.sample</code>
Accuracy measures	<code>roc.curve</code>	<code>accuracy.meas</code>
Accuracy estimators	<code>ROSE.eval</code>	

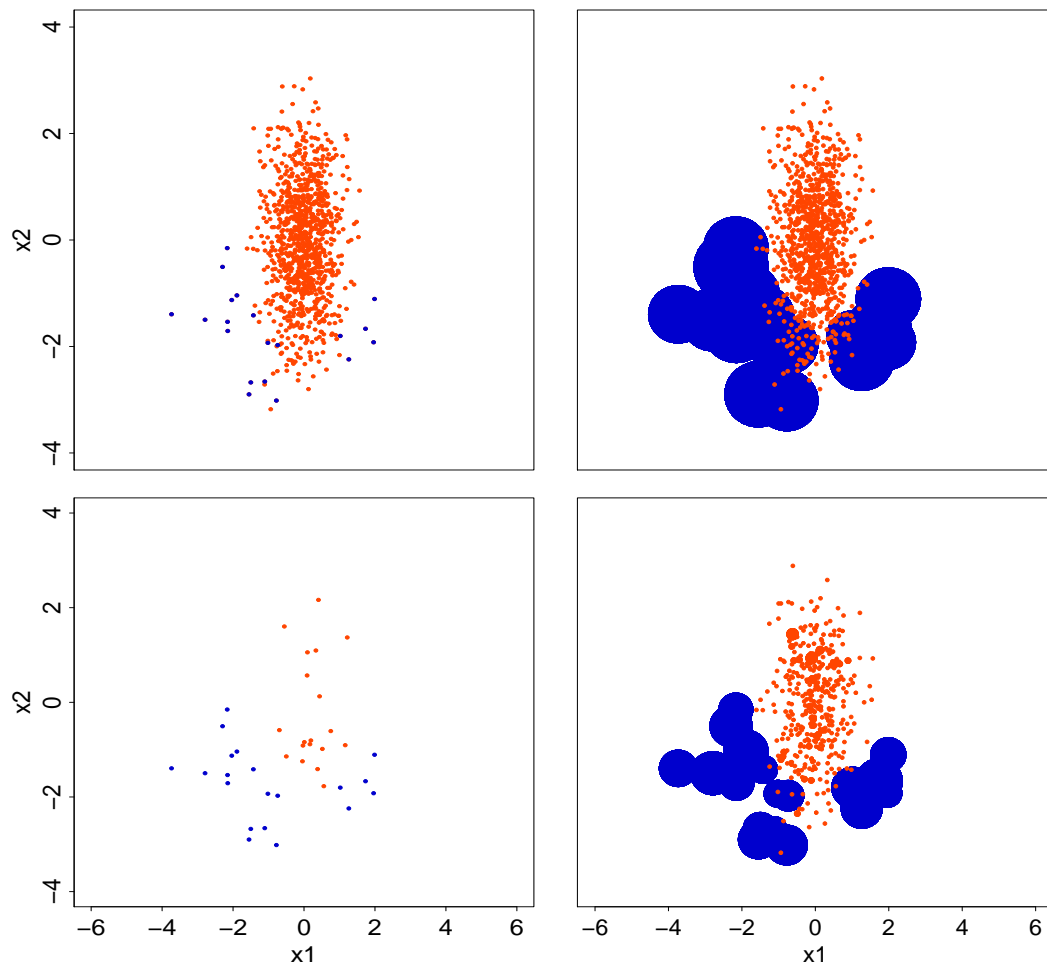


Figure 1: Outcomes of data balancing strategies implemented in the **ROSE** package on the `hacide.train` data. The orange and blue colors denote the majority and minority class examples, respectively. Top-left panel: training data; top-right: oversampling; bottom-left: undersampling; bottom-right: combination of over and undersampling.

```
> data(hacide)

> str(hacide.train)
'data.frame':    1000 obs. of  3 variables:
 $ cls: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
 $ x1 : num  0.2008 0.0166 0.2287 0.1264 0.6008 ...
 $ x2 : num  0.678 1.5766 -0.5595 -0.0938 -0.2984 ...

> table(hacide.train$cls)
 0  1
980 20
```

Now, we show that ignoring class imbalance is not inconsequential. Suppose we want to build a binary classification tree, based on the training data. We first load package `rpart`.

```
> library(rpart)
> treeimb <- rpart(cls ~ ., data = hacide.train)
```

In the current example, the accuracy of the estimated classifier may be evaluated by a standard application of the holdout method, because as many data as we need can be simulated from the same distribution as the training sample to test the classifier. For the sake of illustration, a test sample `hacide.test` is supplied by the package.

Irrespective of the performance metric used for evaluation, a prediction on the test data is first required.

```
> pred.treeimb <- predict(treeimb, newdata = hacide.test)
> head(pred.treeimb)
      0      1
1 0.9898888 0.0101122
2 0.9898888 0.0101122
3 0.9898888 0.0101122
4 0.9898888 0.0101122
5 0.9898888 0.0101122
6 0.9898888 0.0101122
```

The accuracy may be now assessed by means of the performance metrics `accuracy.meas` or `roc.curve` provided by the package. These functions share the mandatory arguments `response` and `predicted`, representing true class labels and the predictions of the classifier, respectively. Predicted values may take the form of a vector of class labels, or alternatively may represent the probability or some score of belonging to the positive class.

```
> accuracy.meas(hacide.test$cls, pred.treeimb[,2])
```

Call:

```
accuracy.meas(response = hacide.test$cls, predicted = pred.treeimb[,2])
```

Examples are labelled as positive when predicted is greater than 0.5

```
precision: 1.000
recall: 0.200
F: 0.167
```

Function `accuracy.meas` computes recall, precision, and the F measure. The estimated classifier shows maximum precision, that is, there are no false positives. On the other hand, recall is very low, thereby implying that the model has predicted a large number of false negatives.

Function `accuracy.meas` is endowed with an optional argument `threshold`, which defines the predicted value over which an example is assigned to the rare class. As one can deduce by the the output above, argument `threshold` defaults to 0.5, like the standard cut-off probability adopted in balanced learning. In the current example, moving `threshold` does not improve classification (results not reported) but this is usually an advisable practice in imbalanced learning. Indeed, the default choice is often too high and might lead to not labeling any example as positive, which would entail undefined values for precision, recall, and F.

To safeguard the user by an arbitrary specification of `threshold` in `accuracy.meas`, the package supplies function `roc.curve`, which computes the area under the ROC curve (AUC) as a measure of accuracy and is not affected by the choice of any particular cut-off value.

```
> roc.curve(hacide.test$cls, pred.treeimb[,2], plotit = FALSE)
Area under the curve (AUC): 0.600
```

Additionally, when optional argument `plotit` is left to its default `TRUE`, `roc.curve` makes an internal call to `plot` and displays the ROC curve in a new window. Further arguments of functions `plot` and `lines` can be invoked in `roc.curve` to customize the resulting ROC curve.

In the current example, the returned AUC is small, thereby indicating that the poor prediction is due to the class imbalance and is not imputable to a wrong threshold.

Balancing the data

The example above highlights the need of adopting a cure for the class imbalance. The first-aid set of remedies provided by the package involves creation of a new artificial data set by suitably resampling the observations belonging to the two classes. Function `ovun.sample` embeds some consolidated resampling techniques to perform such a task and considers different sampling schemes. It is endowed with the argument `method`, which takes one value among `c("over", "under", "both")`.

Option `"over"` determines simple oversampling with replacement from the minority class until either the specified sample size `N` is reached or the positive examples have probability `p` of occurrence. Thus, when `method = "over"`, an augmented sample is returned. Since the prevalent class amounts to 980 observations, to obtain a balanced sample by oversampling, we need to set the new sample size to 1960.

```
> data.bal.ov.N <- ovun.sample(cls ~ ., data = hacide.train, method = "over",
                               N = 1960)$data
> table(data.bal.ov.N$cls)
  0   1
980 980
```

Function `ovun.sample` returns an object of class `list` whose elements are the matched call, the method for data balancing, and the new set of balanced data, which has been directly extracted here. Alternatively, we may design the oversampling by setting argument `p`, which represents the probability of the positive class in the new augmented sample. In this case, the proportion of positive examples will be only approximatively equal to the specified `p`.

```
> data.bal.ov.p <- ovun.sample(cls ~ ., data = hacide.train, method = "over",
                               p = 0.5)$data
> table(data.bal.ov.p$cls)
  0   1
980 986
```

In general, a reader who executes this code would obtain a different distribution of the two classes, because of the randomness of the data generation. To keep trace of the generated sample, a seed may be specified:

```
> data.bal.ov <- ovun.sample(cls ~ ., data = hacide.train, method = "over",
                             p = 0.5, seed = 1)$data
```

The code chunks above also show how to instruct function `ovun.sample` to recognize the different roles of the column data, namely through the specification of the first argument formula. This expects the response variable `cls` on the left-hand side and the predictors on the right-hand side, in the guise of most R regression and classification routines. As usual, the `'.'` has to be interpreted as 'all columns not otherwise in the formula'.

Similar to option "over", option "under" determines simple undersampling without replacement of the majority class until either the specified sample size `N` is reached or the positive examples has probability `p` of occurring. It then turns out that when `method = "under"`, a sample of reduced size is returned. For example, if we set `p`, then

```
> data.bal.un <- ovun.sample(cls ~ ., data = hacide.train, method = "under",
                             p = 0.5, seed = 1)$data
> table(data.bal.un$cls)
  0   1
19 20
```

When `method = "both"` is selected, both the minority class is oversampled with replacement and the majority class is undersampled without replacement. In this case, both the arguments `N` and `p` have to be set to establish the amount of oversampling and undersampling. Essentially, the minority class is oversampled to reach a size determined as a realization of a binomial random variable with size `N` and probability `p`. Undersampling is then performed accordingly, to abide by the specified `N`.

```
> data.bal.ou <- ovun.sample(cls ~ ., data = hacide.train, method = "both",
                             N = 1000, p = 0.5, seed = 1)$data
> table(data.bal.ou$cls)
  0   1
520 480
```

From a qualitative viewpoint, these strategies produce rather different artificial data sets. A flavor of these differences is illustrated in Figure 1, where the outcome of running the three options of function `ovun.sample` on data `hacide.train` is displayed. Each observation appearing in the resulting balanced data set is represented by a point whose size is proportional to the number of ties. Oversampling produces a considerable amount of repeated observations among the rare examples, while undersampling excludes a large number of observations from the prevalent class. A combination of over- and undersampling is a compromise between the two, but still produces several ties for the minority examples when the original training set size is large and the imbalance is extreme.

Data generation according to ROSE attempts to circumvent the pitfalls of the resampling methods above by drawing a new, synthetic, possibly balanced, set of data from the two conditional kernel density estimates of the classes. Endowed with arguments `formula`, `data`, `N`, and `p`, function `ROSE` shares most of its usage with `ovun.sample`:

```
> data.rose <- ROSE(cls ~ ., data = hacide.train, seed = 1)$data
```

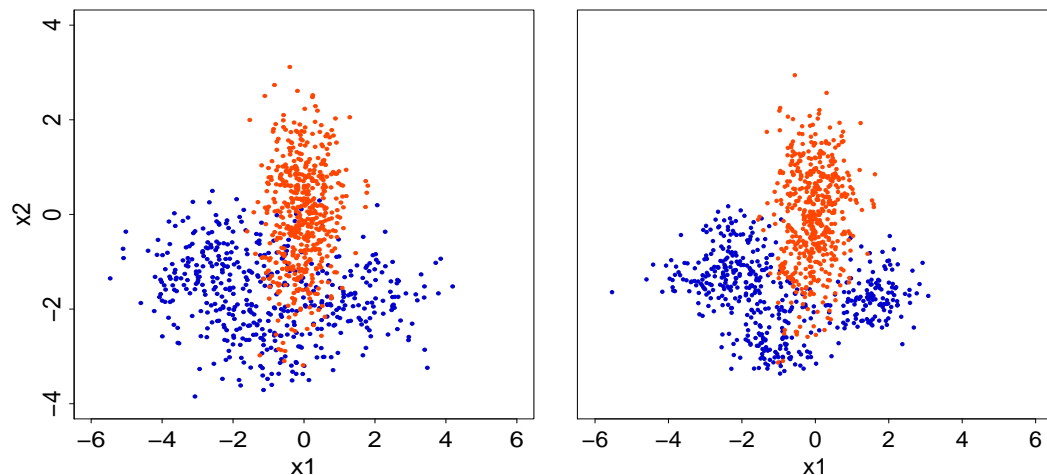


Figure 2: Illustration of `hacide.train` data balanced by ROSE. Left panel: all arguments of function ROSE have been set to the default values. Right panel: smoothing parameters have been shrunk, by setting `hmult.majo = 0.25` and `hmult.mino = 0.5`.

The optional argument `seed`, has been specified here only for reproducibility, while additional arguments have been left to their defaults. In particular, the size N of the new artificial sample is set by default to the size of the input training data, and $p = 0.5$ produces an approximately balanced sample:

```
> table(data.rose$cls)
  0  1 
520 480
```

Figure 2 shows that, unlike the simple balancing mechanism provided by `ovun.sample`, ROSE generation does not produce ties and it actually provides the learner with the option of enlarging the neighborhoods of the original feature space when generating new observations. The widths of such neighborhoods, governed by the matrices \mathbf{H}_0 and \mathbf{H}_1 , are primarily selected as asymptotically optimal under the assumption that the true conditional densities underlying the data follow a Normal distribution (see [Menardi and Torelli, 2014](#), for further details). However, \mathbf{H}_0 and \mathbf{H}_1 may be scaled by arguments `hmult.majo` and `hmult.mino`, respectively, whose default values are set to 1. Smaller (larger) values of these arguments have the effect of shrinking (inflating) the entries of the corresponding smoothing matrix \mathbf{H}_j . Shrinking would be a cautious choice if there is a concern that excessively large neighborhoods could lead to blur the boundaries between the regions of the feature space associated with each class. For example, we could set

```
> data.rose.h <- ROSE(cls ~ ., data = hacide.train, seed = 1, hmult.majo = 0.25,
  hmult.mino = 0.5)$data
```

The generated data are illustrated in the right panel of Figure 2. To better understand the rationale behind ROSE, Figure 3 (left panel) shows two observations belonging to the prevalent and rare classes, and the surroundings within the observations are likely to be generated, for the two options `hmult.majo = 0.25, hmult.mino = 0.5`.

Equipped with the new balanced samples generated by `ovun.sample` and ROSE, we can train the classifiers and run them on the test set to assess their accuracy:

```
> tree.rose <- rpart(cls ~ ., data = data.rose)
> tree.ov <- rpart(cls ~ ., data = data.bal.ov)
> tree.un <- rpart(cls ~ ., data = data.bal.un)
> tree.ou <- rpart(cls ~ ., data = data.bal.ou)

> pred.tree.rose <- predict(tree.rose, newdata = hacide.test)
> pred.tree.ov <- predict(tree.ov, newdata = hacide.test)
> pred.tree.un <- predict(tree.un, newdata = hacide.test)
> pred.tree.ou <- predict(tree.ou, newdata = hacide.test)

> roc.curve(hacide.test$cls, pred.tree.rose[,2])
Area under the curve (AUC): 0.989
```

```
> roc.curve(hacide.test$cls, pred.tree.ov[,2], add.roc = TRUE, col = 2, lty = 2)
Area under the curve (AUC): 0.798
> roc.curve(hacide.test$cls, pred.tree.un[,2], add.roc = TRUE, col = 3, lty = 3)
Area under the curve (AUC): 0.749
> roc.curve(hacide.test$cls, pred.tree.ou[,2], add.roc = TRUE, col = 4, lty = 4)
Area under the curve (AUC): 0.798
```

The returned AUCs clearly highlight the effectiveness of balancing the sample. An illustration of the ROC curves is given in the right panel of Figure 3.

Adopting alternative estimators of accuracy

In real data applications, we often cannot benefit from the availability of additional data to test the accuracy of the estimated model (or if we can, we will probably use the additional information to train a more accurate model). Moreover, in imbalanced learning, the scarcity of data causes high variance estimates of the accuracy measure. Then, it is often appropriate to adopt some alternative methods to estimate model accuracy in place of the standard holdout. Function `ROSE.eval` comes to this aid by implementing a ROSE version of holdout, bootstrap or leave-K-out cross-validation estimators of the accuracy of a specified classifier, as measured according to a selected metric.

Suppose that a test set, to evaluate the accuracy of the classifier previously denoted as `tree.rose`, is not available. We first exploit `ROSE.eval` to obtain a ROSE-based holdout estimate of the AUC:

```
> ROSE.holdout <- ROSE.eval(cls ~ ., data = hacide.train, learner = rpart,
  method.assess = "holdout", extr.pred = function(obj)obj[,2], seed = 1)
> ROSE.holdout
```

Holdout estimate of auc: 0.985

Similarly to ROSE, the former instruction specifies the role of the column data from which a ROSE sample is generated via argument `formula`. The specification of `seed` ensures the generation of the same sample as the one used to build `tree.rose`. Based on the synthetic sample, the classifier is built through an internal call to function `rpart` specified in `learner`.

Argument `learner` is designed to recognize virtually all the R functions having a standard definition, as well as suitably user-defined functions, thereby allowing for a high flexibility in the employment of classifiers which exhibit non-homogeneous usage. As a consequence of such heterogeneity, some caution is needed when specifying argument `learner`. In its simplest form, `learner` is a standard R function, that is, it has a formula as a first compulsory argument and returns an object

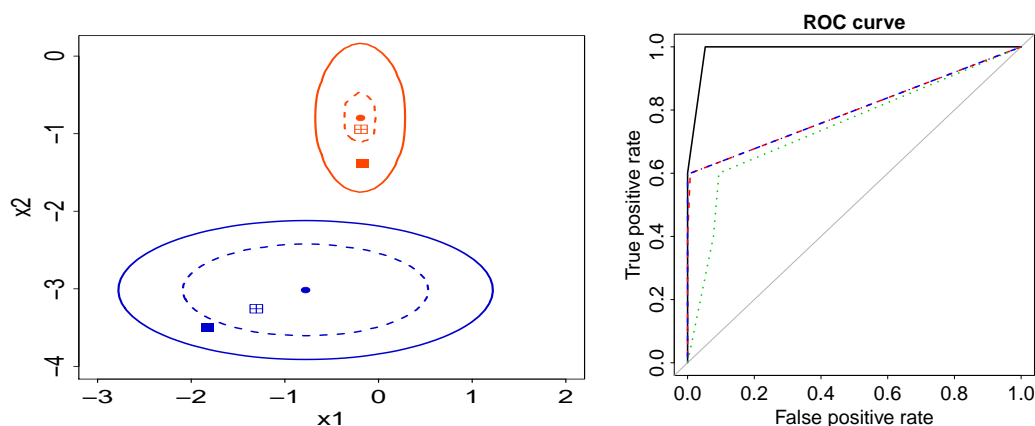


Figure 3: Left panel: a zoomed view of two selected points of the `hacide.train` data, the contour within which the ROSE artificial data are generated with probability 0.95 and an example of artificial points generated. The solid contour (and the associated filled points) refer to the use of the default smoothing parameters. The dashed contour (and associated points filled with a cross) refer to the use of `hmult.majo = 0.25` and `hmult.mino = 0.5`. Right panel: ROC curves associated with classification trees built on data balanced according to different remedies. Solid line: data balanced by ROSE; dotted line: data balanced by undersampling; dashed (overlapping) lines: data balanced by oversampling and by a combination of over and undersampling.

whose class is associated with a predict method. As required by R conventions, the function name of the classifier and its R class should match. For example, in the chunk code above, `learner = rpart` is specified, as function `rpart` returns an object of class `rpart` to which a `predict.rpart` method is associated. Thus, function `predict.rpart` is internally called by `ROSE.eval` to obtain a prediction. Also classifiers implemented by R functions with a non-standard usage can be passed to `learner`. In this case, `learner` takes a different form, as will be clarified subsequently in this section.

Function `ROSE.eval` works properly only when the prediction is a vector of class labels, probabilities, or scores from the estimated learner. Since predict methods in R sources exhibit very non-homogeneous outputs, the optional argument `extr.pred` can be deployed to pass a suitable user-defined function that filters relevant information from the output of any predict method and extracts a vector of predicted values. In the example above, `predict.rpart` returns a matrix of predicted probabilities when fed with a classification tree. Hence, argument `extr.pred = function(obj) obj[,2]` is defined as a function which extracts the column of probabilities of belonging to the positive class.

Along with the argument `learner`, function `ROSE.eval` requires the specification of argument `acc.measure`, an accuracy measure to be selected among `c("auc", "precision", "recall", "F")`, which are the metrics supplied by the package. In the current example, `acc.measure` has been left to its default value `"auc"`, which determines an internal call to function `roc.curve`. The other options entail an internal call to function `accuracy.meas`.

As an alternative to the ROSE-based holdout method, we may wish to obtain a ROSE version of the bootstrap accuracy distribution. By selecting `method.assess = "BOOT"`, we fit the specified learner on `B` ROSE samples and test each of them on the observed data specified in the formula. The optional argument `trace` shows the progress of model assessment by printing the number of completed iterations. The bootstrap distribution of the selected accuracy measure is then returned as output of function `ROSE.eval`.

```
> ROSE.boot <- ROSE.eval(cls ~ ., data = hacide.train, learner = rpart,
  method.assess = "BOOT", B = 50, extr.pred = function(obj) obj[,2],
  seed = 1, trace = TRUE)
```

Iteration:

10, 20, 30, 40, 50

```
> summary(ROSE.boot)
```

Call:

```
ROSE.eval(formula = cls ~ ., data = hacide.train, method.assess = "BOOT",
  B = 50, learner = rpart, extr.pred = function(obj) obj[, 2],
  trace = TRUE, seed = 1)
```

Summary of bootstrap distribution of auc:

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.9282	0.9800	0.9839	0.9820	0.9860	0.9911

Similarly, we could obtain a leave-K-out cross validation estimate of the accuracy by selecting option `method.assess = "LKOCV"`. This option works as follows: the provided data, say of size n , are split at random into $Q = n/K$ groups of size defined by argument `K`; to speed up the computations, we set `K = 5`. However, in the presence of extreme imbalance between the classes, choosing `K` as low as possible is advisable, to not alter the class distribution in the data used for training. At each round, the specified learner is estimated on a ROSE sample built on the provided data excluding one of these groups, and then a prediction on the excluded set of observations is made. At the end of the process, the predictions obtained on the Q distinct groups are deployed to compute the selected accuracy measure, which is then retrieved in the component `acc` of the output of `ROSE.eval`.

```
> ROSE.l5ocv <- ROSE.eval(cls ~ ., data = hacide.train, learner = rpart,
  method.assess = "LKOCV", K = 5, extr.pred = function(obj) obj[,2],
  seed = 1, trace = TRUE)
```

Iteration:

10, 20, 30, 40, 50, 60, 70, 80, 90, 100,
110, 120, 130, 140, 150, 160, 170, 180, 190, 200

```
> ROSE.l5ocv
```

Call:

```
ROSE.eval(formula = cls ~ ., data = hacide.train, learner = rpart,
  extr.pred = function(obj) obj[, 2], method.assess = "LKOCV",
  K = 5, trace = TRUE, seed = 1)
```

Leave K out cross-validation estimate of auc: 0.973

In principle, the classification task would terminate here. However, one might wonder if different choices in the balancing mechanism or in the classifier estimation would lead to better results. To answer this question, we just need to set arguments `control.ROSE` or, respectively `control.learner` accordingly. For example the `hmult.majo` and/or `hmult.mino` arguments could have been shrunk:

```
> ROSE.holdout.h <- ROSE.eval(cls ~ ., data = hacide.train, learner = rpart,
  extr.pred = function(obj)obj[,2], seed = 1,
  control.rose = list(hmult.majo = 0.25, hmult.mino = 0.5))
```

or one could have grown a tree of a given complexity, in the following manner:

```
> ROSE.holdout.cp <- ROSE.eval(cls ~ ., data = hacide.train, learner = rpart,
  extr.pred = function(obj)obj[,2], seed = 1,
  control.learner = list(control=list(cp = 0.05)))
```

Lastly, one could wish to compare the results with those obtained by using a different classifier. Suppose, for example, one is interested in fitting a K -nearest neighbors classifier. This is implemented, among others, by function `knn` in package `class`. Function `knn` has a nonstandard R behaviour, both because it does not require a formula argument and because model training and prediction are performed, by construction, simultaneously. The usage of function `knn` is described by

```
knn(train, test, cl, k = 1, l = 0, prob = FALSE, use.all = TRUE)
```

where `train` and `test` are the training and the testing columns of predictors, respectively, and `cl` is the vector of training class labels. To employ `knn` as a learner function in `ROSE.eval`, we need to embed it within a suitable wrapper function which requires two mandatory arguments: `data` and `newdata`. The wrapper for `knn` is then:

```
knn.wrap <- function(data, newdata, ...){
  knn(train = data[, -1], test = newdata, cl = data[, 1], ...)}
```

where the first column is extracted as, in the current `hacide` example, it contains the class labels. Arguments `data` and `newdata` are automatically filled when the wrapper is embedded in `ROSE.eval`. Further arguments can be passed to `knn` through the usual argument `control.learner`, owing to the presence of argument dots in the wrapper. For example, we may specify $k = 2$ nearest neighbors and ask to return the predicted probabilities of belonging to the positive class as follows.

```
> rose.holdout.knn <- ROSE.eval(cls ~ ., data = hacide.train, learner = knn.wrap,
  control.learner = list(k = 2, prob = TRUE),
  method.assess = "holdout", seed = 1)
```

As we can see, even if the employed classifier has a nonstandard implementation, it can be easily adapted to be used inside function `ROSE.eval`.

Bibliography

- A. W. Bowman and A. Azzalini. *Applied Smoothing Techniques for Data Analysis: Kernel Approach with S-Plus Illustrations*. Oxford University Press, Oxford, 1997. [p80]
- B. Efron and R. Tibshirani. *An Introduction to the Bootstrap*. Chapman and Hall, London, 1993. [p80]
- M. Kuhn. *caret: Classification and Regression Training*, 2014. URL <http://CRAN.R-project.org/package=caret>. R package version 6.0-22. Contributions from Jed Wing and Steve Weston and Andre Williams and Chris Keefer and Allan Engelhardt and Tony Cooper and Zachary Mayer and the R Core Team. [p79]
- N. Lunardon, G. Menardi, and N. Torelli. *R package ROSE: Random Over-Sampling Examples (version 0.0-3)*. Università di Trieste and Università di Padova, Italia, 2013. URL <http://cran.r-project.org/web/packages/ROSE/index.html>. [p79]
- G. Menardi and N. Torelli. Training and assessing classification rules with imbalanced data. *Data Mining and Knowledge Discovery*, 28(1):92–122, 2014. [p79, 80, 81, 85]
- L. Torgo. *Data Mining with R, Learning with Case Studies*. Chapman and Hall/CRC, London, 2010. URL <http://www.dcc.fc.up.pt/~ltorgo/DataMiningWithR>. [p79]

Nicola Lunardon

*Department of Economics, Business, Mathematics, and Statistics “Bruno de Finetti”, University of Trieste
Piazzale Europa 1, 34100 Trieste, Italy*

lunardon@stat.unipd.it

Giovanna Menardi

*Department of Statistical Sciences, University of Padua
via Cesare Battisti, 241, 35100 Padova, Italy*

menardi@stat.unipd.it

Nicola Torelli

*Department of Economics, Business, Mathematics, and Statistics “Bruno de Finetti”, University of Trieste
Piazzale Europa 1, 34100 Trieste, Italy*

nicola.torelli@econ.units.it

investr: An R Package for Inverse Estimation

by Brandon M. Greenwell and Christine M. Schubert Kabban

Abstract Inverse estimation is a classical and well-known problem in regression. In simple terms, it involves the use of an observed value of the response to make inference on the corresponding unknown value of the explanatory variable. To our knowledge, however, statistical software is somewhat lacking the capabilities for analyzing these types of problems. In this paper, we introduce **investr** (which stands for **inverse estimation in R**), a package for solving inverse estimation problems in both linear and nonlinear regression models.¹

Introduction

Consider the regression model $\mathcal{Y}_i = f(x_i; \beta) + \epsilon_i$ ($i = 1, \dots, n$), where f is a known expectation function (called a *calibration curve*) that is monotonic over the range of interest and $\epsilon_i \stackrel{iid}{\sim} \mathcal{N}(0, \sigma^2)$. A common problem in regression is to predict a future response \mathcal{Y}_0 from a known value of the explanatory variable x_0 . Often, however, there is a need to do the reverse; that is, given an observed value of the response $\mathcal{Y} = y_0$, estimate the unknown value of the explanatory variable x_0 . This is known as the *calibration problem*, though we refer to it more generally as inverse estimation. In this paper, we consider only *controlled calibration*, where the values of the explanatory variables are fixed by design. A more thorough overview of the calibration problem, including Bayesian approaches and multivariate calibration, is given in Osborne (1991).

There are three main functions available in the **investr** package (Greenwell, 2014):

- `calibrate`;
- `invest`;
- `plotFit`.

`calibrate` operates on objects of class "lm" and can only be used when the expectation function has the form $f(x_i; \beta) = \beta_0 + \beta_1 x_i$ (i.e., the simple linear regression model), where closed-form solutions are available for calculating confidence intervals for x_0 . For more complicated models (e.g., polynomial and nonlinear regression), closed-form expressions are usually not available and iterative techniques will be required. This is the purpose of the function `invest`, which calculates the point estimate and confidence intervals for x_0 by calling the function `uniroot` from the **stats** package. The function `plotFit` produces a scatterplot of the data with fitted regression curve and the option to add confidence/prediction bands for the response (pointwise or adjusted). It can be used with single-predictor objects of class "lm" or "nls"; however, for objects of class "nls", confidence/prediction bands are based on the linear approximation and can be misleading (Bates and Watts, 1988, p. 65). The development version of **investr** can be found on GitHub at <https://github.com/w108bmg/investr>. To report bugs or issues, contact the main author or submit them to <https://github.com/w108bmg/investr/issues>.

Calibration for straight line regression

Consider the most common calibration model, $\mathcal{Y}_i = \beta_0 + \beta_1 x_i + \epsilon_i$ ($i = 1, \dots, n$), where x_i is fixed and $\epsilon_i \stackrel{iid}{\sim} \mathcal{N}(0, \sigma^2)$. Suppose we obtain a series of m observations y_{01}, \dots, y_{0m} which are associated with the same (unknown) x_0 . It can be shown (Graybill, 1976) that the maximum likelihood (ML) estimator of x_0 , also known as the *classical estimator*, is

$$\hat{x}_0 = \frac{\bar{y}_0 - \hat{\beta}_0}{\hat{\beta}_1}, \quad (1)$$

where $\hat{\beta}_0$ and $\hat{\beta}_1$ are the usual ML estimators of β_0 and β_1 , respectively, and $\bar{y}_0 = \sum_{j=1}^m y_{0j} / m$. The classical estimator in general is computed as $\hat{x}_0 = f^{-1}(\bar{y}_0; \hat{\beta})$; that is, by inverting the fitted calibration curve at \bar{y}_0 (Eisenhart, 1939). Since \hat{x}_0 is a ratio of jointly normal random variables, it is not

¹The views expressed in this paper are those of the authors, and do not reflect the official policy or position of the United States Air Force, Navy, Department of Defense, or the U.S. Government.

surprising to learn that it does not have any finite moments (think of a standard Cauchy distribution). The sampling distribution of \hat{x}_0 is complicated, but fortunately, not required for setting an exact $100(1 - \alpha)\%$ confidence interval on x_0 . This is discussed in the next section.

Inversion interval

It can be shown (see, for example, [Draper and Smith, 1998](#), pp. 47–51) that an exact $100(1 - \alpha)\%$ confidence interval for x_0 is given by

$$\hat{x}_0 + \frac{(\hat{x}_0 - \bar{x})g \pm (t\hat{\sigma}/\hat{\beta}_1) \sqrt{(\hat{x}_0 - \bar{x})^2/S_{xx} + (1-g)\left(\frac{1}{m} + \frac{1}{n}\right)}}{1-g}, \quad (2)$$

where $S_{xx} = \sum_{i=1}^n (x_i - \bar{x})^2$, $g = (t^2\hat{\sigma}^2) / (\hat{\beta}_1^2 S_{xx})$ and $t = t_{\alpha/2, n+m-3}$ is the upper $1 - \alpha$ percentile of a Student's t -distribution with $n + m - 3$ degrees of freedom. The inversion interval (2) can also be obtained using a fiducial argument, as in [Fieller \(1954\)](#). For the special case $m = 1$, the inversion interval is equivalent to inverting a $100(1 - \alpha)\%$ prediction interval for the response. In other words, if one draws a horizontal line through a scatterplot of the data at y_0 , then the abscissas of its intersection with the usual (pointwise) prediction band for f correspond to the endpoints of the inversion interval (2). This interval should only be used when the usual F test for testing $\mathcal{H}_0 : \beta_1 = 0$ versus $\mathcal{H}_1 : \beta_1 \neq 0$ can be rejected at the specified α level; in other words, when the regression line is not “too” flat. If \mathcal{H}_0 is not rejected, then such a confidence interval for x_0 may result in either the entire real line or two semi-infinite intervals ([Graybill and Iyer, 1994](#), p. 429)—see Figure 2. The `plotFit` function in the **investr** package can be used for drawing scatterplots with the fitted model and confidence/prediction bands. To calculate the inversion interval for the linear calibration problem, we use the `calibrate` function and specify the option `interval = "inversion"` (the default) as in the following example.

The data frame `arsenic` contains the true amounts of arsenic present in 32 water samples ([Graybill and Iyer, 1994](#)). Also present, is the amount of arsenic measured by some field test, which is subject to error. A new water sample is obtained and subjected to the field test producing a reading of $3.0 \mu\text{g/ml}$. It is desired to infer the true amount of arsenic present in the sample. The following code fits a simple linear regression model to the arsenic data and then uses the `calibrate` function to compute the ML estimate and corresponding 90% calibration interval based on Equation (2).

```
library(investr)
mod <- lm(measured ~ actual, data = arsenic)
(res <- calibrate(mod, y0 = 3, interval = "inversion", level = 0.9))

## Figure 1
plotFit(mod, interval = "prediction", level = 0.9, shade = TRUE, col.pred = "skyblue")
abline(h = 3, v = c(res$lower, res$estimate, res$upper), lty = 2)
```

Running the above block of code produces Figure 1 and the following output to the R console:

```
estimate    lower    upper
   2.9314    2.6035    3.2587
```

where `estimate` is the ML estimate (1), and `lower/upper` correspond to the lower/upper bounds of the 90% inversion interval for x_0 (2). If instead the new water sample was subjected to the field test three times, thereby producing three response values corresponding to x_0 , say 3.17, 3.09, and 3.16 $\mu\text{g/ml}$, we would simply supply `calibrate` with a vector of these values as in

```
calibrate(mod, y0 = c(3.17, 3.09, 3.16), interval = "inversion", level = 0.9)
```

If `interval = "inversion"`, and the slope of the model is not significant at the specified α level, then finite confidence limits for x_0 will not be produced. For example, suppose `badfit` is an “lm” object for which the slope is not significant at the $\alpha = 0.1$ level. Then, as illustrated in Figure 2,

```
calibrate(badfit, y0 = 10, level = 0.9)
```

will either produce two semi-infinite intervals, e.g.,

```
Error: The calibration line is not well determined. The resulting
confidence region is the union of two semi-infinite intervals:
( -Inf , -282.0006 ) U ( 393.1267 , Inf)
```

or the entire real line, e.g.,

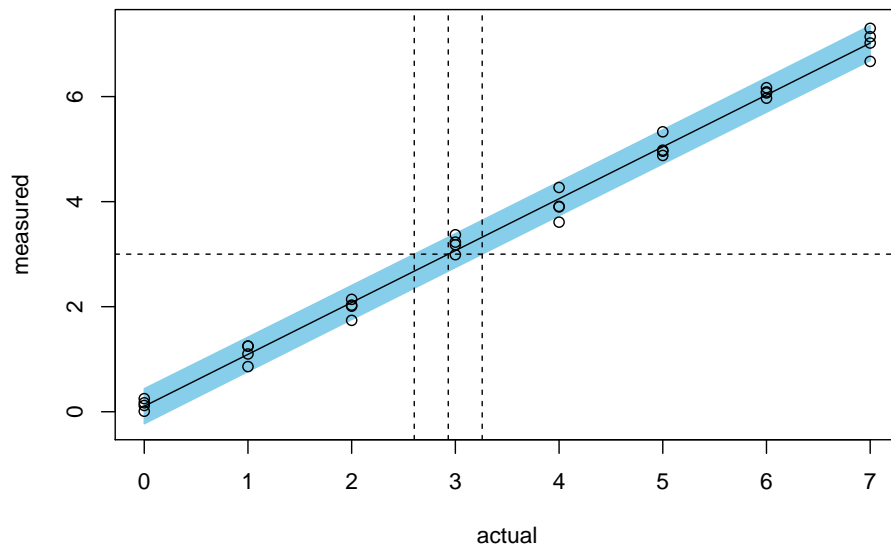


Figure 1: Scatterplot of the arsenic data with fitted calibration line and pointwise prediction band. A horizontal reference line is drawn through the observed value $y_0 = 3$. The vertical lines identify the position of the point estimate and 90% confidence bounds for x_0 .

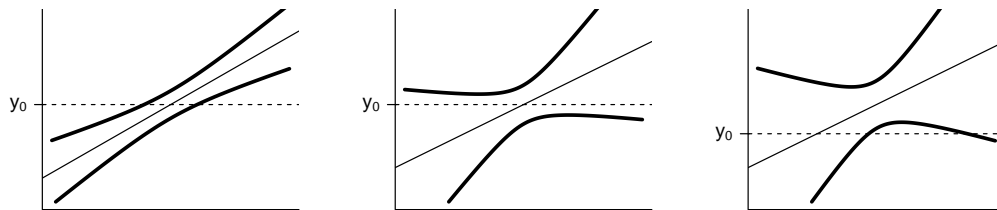


Figure 2: Hypothetical $1 - \alpha$ (pointwise) prediction bands. *Left:* Horizontal line at y_0 intersects the prediction band at two points, resulting in a finite interval. *Middle:* Horizontal line at y_0 does not intersect the prediction band at all resulting in an infinite interval. *Right:* Horizontal line at y_0 only intersects with one side of the prediction band resulting in two semi-infinite intervals.

```
estimate    lower    upper
-97.5987   -Inf     Inf
Warning message:
The calibration line is not well determined.
```

Wald interval

Another common approach to computing calibration intervals is to use the delta method (Dorfman, 1938). It is easy to show that an approximate standard error for the ML estimator (1), based on a first-order Taylor series approximation, is given by

$$\widehat{\text{se}}(\hat{x}_0) = \frac{\hat{\sigma}}{|\hat{\beta}_1|} \sqrt{\frac{1}{m} + \frac{1}{n} + \frac{(\hat{x}_0 - \bar{x})^2}{S_{xx}}}. \quad (3)$$

Assuming large sample normality for \hat{x}_0 leads to an approximate $100(1 - \alpha)\%$ Wald confidence interval for x_0 of

$$\hat{x}_0 \pm t_{\alpha/2, n+m-3} \frac{\hat{\sigma}}{|\hat{\beta}_1|} \sqrt{\frac{1}{m} + \frac{1}{n} + \frac{(\hat{x}_0 - \bar{x})^2}{S_{xx}}}. \quad (4)$$

This is equivalent to taking $g = 0$ in the inversion interval (2). Unlike the inversion interval, though, the Wald interval always exists and is symmetric about \hat{x}_0 . This symmetry is attractive, but not always realistic, such as when the calibration curve f is nonlinear and has horizontal asymptotes. To obtain a Wald-type interval we specify interval = "Wald" in the call to calibrate. For instance,

```
calibrate(mod, y0 = 3, interval = "Wald", level = 0.9)
```

produces the output

```
estimate    lower    upper    se
  2.9314    2.6040    3.2589    0.1929
```

The point estimate remains unchanged (as expected) but the calibration interval is slightly different. The major benefit of using the delta method to compute calibration intervals is that it always exists and provides us with an asymptotic estimate of the standard error, which in this case gives $\hat{se} = 0.1929$. The bootstrap, as discussed in Section [Bootstrap calibration intervals](#), also provides calibration intervals and an estimate of the standard error, but does not require large samples or specific distribution assumptions.

Confidence interval for a specified mean response

Instead of inferring the value of x_0 that corresponds to an observed value of the response y_0 , suppose we want to infer the value of x_0 that corresponds to a specified value of the mean response, say $f(x_0; \beta) = \mu_0$. The obvious ML estimate of x_0 is

$$\hat{x}_0 = \frac{\mu_0 - \hat{\beta}_0}{\hat{\beta}_1}, \quad (5)$$

which is the same as Equation (1) but with \bar{y}_0 replaced with μ_0 , a fixed population parameter. This is analogous to the difference between (i) predicting a future value of the response corresponding to a given value of the explanatory variable and (ii) estimating the mean response that corresponds to a particular value of the explanatory variable. Thus, the point estimates (1) and (5) are the same but the former has greater variability inherited from the variance of \bar{Y}_0 . This is sometimes referred to as *regulation* (as opposed to calibration) and is described in more detail in [Graybill and Iyer \(1994, Chap. 6\)](#). The confidence interval formulas for x_0 corresponding to a specified mean response (i.e., regulation) are the same as those given in Equations (2) and (4), but with $1/m$ replaced with 0 and $t_{\alpha/2, n+m-3}$ replaced with $t_{\alpha/2, n-2}$. For instance, the Wald interval for x_0 corresponding to μ_0 is simply

$$\hat{x}_0 \pm t_{\alpha/2, n-2} \frac{\hat{\sigma}}{|\hat{\beta}_1|} \sqrt{\frac{1}{n} + \frac{(\hat{x}_0 - \bar{x})^2}{S_{xx}}},$$

where \hat{x}_0 is calculated as in Equation (5). To obtain calibration intervals corresponding to a specified mean response, use the option `mean.response = TRUE`.

To illustrate, consider the crystal growth data taken from [Graybill and Iyer \(1994, p. 119\)](#). These data are from an experiment in which the weight in grams of 14 crystals were recorded after letting the crystals grow for different (predetermined) amounts of time in hours. The weight of each crystal is plotted against time in Figure 3. Suppose we want to estimate the growth time in hours that corresponds to an average weight of 8 grams; that is, we want to estimate x_0 corresponding to $\mu_0 = 8$. The following code chunk fits a simple linear regression model to the crystal growth data and then computes the ML estimate and a 95% calibration interval for x_0 .

```
mod <- lm(weight ~ time, data = crystal)
(res <- calibrate(mod, y0 = 8, mean.response = TRUE))

## Figure 3
plotFit(mod, interval = "confidence", pch = 19, shade = TRUE, col.conf = "plum",
        extend.range = TRUE)
abline(h = 8, v = c(res$lower, res$estimate, res$upper), lty = 2)
```

The output for this code chunk should be

```
estimate    lower    upper
  15.8882    14.6590    17.1596
```

Thus, in order to produce crystals with an average weight of 8 grams, they should be grown for an estimated 15.8882 hours. A 95% confidence interval for the growth time is (14.6590, 17.1596). Obviously, if `mean.response = TRUE`, then `y0` can only take a single value; otherwise, an error will be displayed as in the following:

```
calibrate(mod, y0 = c(8, 9), mean.response = TRUE)
```

which displays the message

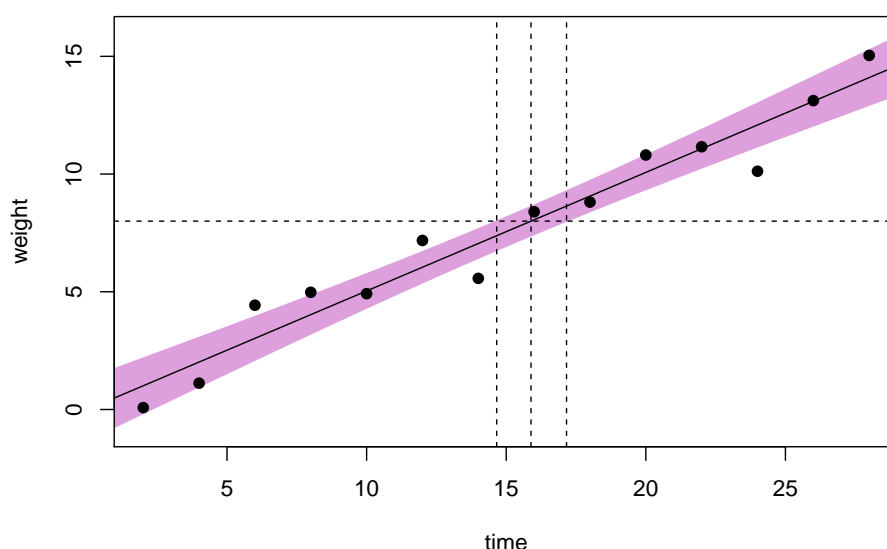


Figure 3: Scatterplot of the crystal growth data with fitted calibration line and pointwise confidence band. A horizontal reference line is drawn at $\mu_0 = 8$. The vertical lines identify the position of the point estimate and 95% confidence bounds for the growth time x_0 .

```
Error in calibrate.default(cbind(x, y), ...) :
  Only one mean response value allowed.
```

This type of calibration problem is similar to computing the *median effective dose* ($ED_{0.5}$), or more generally ED_p where $0 < p < 1$, in binary response models. In R, these models are usually fit with the `glm` function from the **stats** package. The function `dose.p` from the **MASS** package (Venables and Ripley, 2002) can then be used to compute the ML estimate of ED_p for specified p . An estimate of the asymptotic standard error based on the delta method is also given which can be used to calculate a Wald-based confidence interval for ED_p . A future release of **investr** will likely make an inversion-type interval available for these models as well (i.e., by inverting a confidence interval for the mean response on the link scale). The package **drc** (Ritz and Streibig, 2005), for fitting dose-response curves, may also be of interest.

Simultaneous calibration intervals

The calibration intervals discussed so far are one-at-a-time intervals. If k new values of the response are observed where each corresponds to a different unknown, say x_{01}, \dots, x_{0k} , then we need to adjust the critical value used in the inversion and Wald intervals accordingly. The simplest approaches are of course, the *Bonferroni* and *Scheffé* procedures. These can be computed by specifying the `adjust` option which can take any of the following arguments: "none" (the default), "bonferroni", or "scheffe". The value k also needs to be specified. See Miller (1981, Chap. 3) for more details.

Nonlinear and polynomial calibration

In application, many relationships are nonlinear (e.g., dose-response curves). The classical estimator along with the inversion and Wald intervals can easily be extended to such nonlinear calibration curves. However, classical inference in these models (such as prediction intervals) are based on large samples and linear approximations (see Bates and Watts, 1988, Chap. 2). Thus, for nonlinear calibration curves, the inversion interval provides only an approximate $100(1 - \alpha)\%$ confidence interval for x_0 as does the Wald interval. Calibration in nonlinear models is discussed in further detail in Schwenke and Milliken (1991), Seber and Wild (2003, pp. 245–250), and Huet (2004, Chap. 5). For calibration in polynomial models, see Brown (1993, pp. 47–88) and Seber and Lee (2003, p. 172).

The `invest` function can be used for inverse estimation with any univariate regression model in R that inherits from class "lm" or "nls" (with the exception of weighted fits). For instance, consider the regression model $\mathcal{Y}_i = f(x_i; \theta) + \epsilon_i$ ($i = 1, \dots, n$), where f may or may not be linear in θ . If we wish to estimate x_0 given an observation y_0 , then the point estimate \hat{x}_0 is given by solving $y_0 = f(x; \hat{\theta})$ for x . The solution will be unique as long as f is monotonic in the region of interest. The `invest` function

computes \hat{x}_0 by calling the **stats** functions `predict` and `uniroot` to solve

$$y_0 - f(x; \hat{\theta}) = 0$$

numerically for x .

Approximate confidence intervals

Equation (2) gives a closed-form expression for the inversion interval for the case of simple linear regression. In more complicated cases, such an expression is not available and the interval must be found numerically. It can be shown (see, for example, [Seber and Wild, 2003](#), pp. 245–246) that

$$\mathcal{T}_{x_0} = \frac{y_0 - f(x_0; \hat{\theta})}{\{\hat{\sigma}^2 + \hat{S}_{x_0}^2\}^{1/2}} \sim t_{n-p}, \quad (6)$$

where $\hat{S}_{x_0}^2$ is the estimated variance of $f(x_0; \hat{\theta})$. An approximate $100(1 - \alpha)\%$ confidence interval for x_0 is then given by the set

$$\left\{ x : t_{\alpha/2, n-p} < \mathcal{T}_x < t_{1-\alpha/2, n-p} \right\}. \quad (7)$$

Essentially, `invest` finds the lower and upper limits for this interval by solving the equations

$$\mathcal{T}_x - t_{\alpha/2, n-p} = 0 \quad \text{and} \quad \mathcal{T}_x - t_{1-\alpha/2, n-p} = 0$$

numerically for x_0 . For the special case of the simple linear regression model, these limits coincide with Equation (2) and the coverage probability is exactly $1 - \alpha$.

The Wald interval for x_0 is also easily extended. It has the basic form: $\hat{x}_0 \pm t_{\alpha/2, n-p} \hat{\text{se}}(\hat{x}_0)$, where p is the dimension of θ . The estimated standard error $\hat{\text{se}}(\hat{x}_0)$ is based on a first-order Taylor series approximation. For the special case of the simple linear regression model, this approximation results in Equation (3).

Since the point estimate and confidence intervals for x_0 are obtained numerically using `uniroot`, `invest` has the additional options `lower`, `upper`, `tol`, and `maxiter`. See the reference manual for details.

To get an idea of how `invest` works, consider the data in Figure 4 which were generated from the model $\mathcal{Y} = 5 + x - \sin(x) + 1.5Z$, where Z is a standard normal random variable. Suppose we want to estimate x_0 given a new observation, say $y_0 = 22$. Fitting a model of the form $f(x; \theta) = \theta_1 + \theta_2 [x - \sin(x)]$ to the sample, we obtain $\hat{f}(x) = 5.490 + 0.941 [x - \sin(x)]$, which is certainly invertible, but \hat{f}^{-1} cannot be expressed in closed-form using a finite number of terms; thus, $\hat{x}_0 = \hat{f}^{-1}(y_0 = 22)$ must be obtained numerically. The following chunk of code generates the sample data, calculates \hat{x}_0 and an approximate 95% Wald interval for x_0 using the `invest` function.

```
set.seed(101) # for reproducibility
x <- rep(seq(from = 0, to = 25, by = 2), each = 2)
d <- data.frame(x, y = 5 + x - sin(x) + rnorm(length(x), sd = 1.5))
mod <- lm(y ~ I(x - sin(x)), data = d)
res <- invest(mod, y0 = 22, interval = "Wald")

## Figure 4
plotFit(mod, interval = "prediction", shade = TRUE, col.pred = "seagreen1",
        extend.range = TRUE)
abline(h = 22, v = res$estimate, lty = 2)
```

The point estimate is $\hat{x}_0 = 16.7053$ with an estimated standard error of 0.8909. The code used by `invest` for obtaining the point estimate is essentially

```
uniroot(function(x) predict(mod, newdata = list("x" = x)) - 22,
        lower = min(d$x), upper = max(d$x))$root
```

The code for obtaining the standard error used in the Wald interval is slightly more involved:

```
## Write x0.hat as function of parameters (theta1.hat, theta2.hat, Y0)
x0.fun <- function(params, object = mod) {
  object$coefficients <- params[1:2]
  uniroot(function(x) predict(object, list("x" = x)) - params[3],
          lower = 0, upper = 25, tol = 1e-10)$root
```

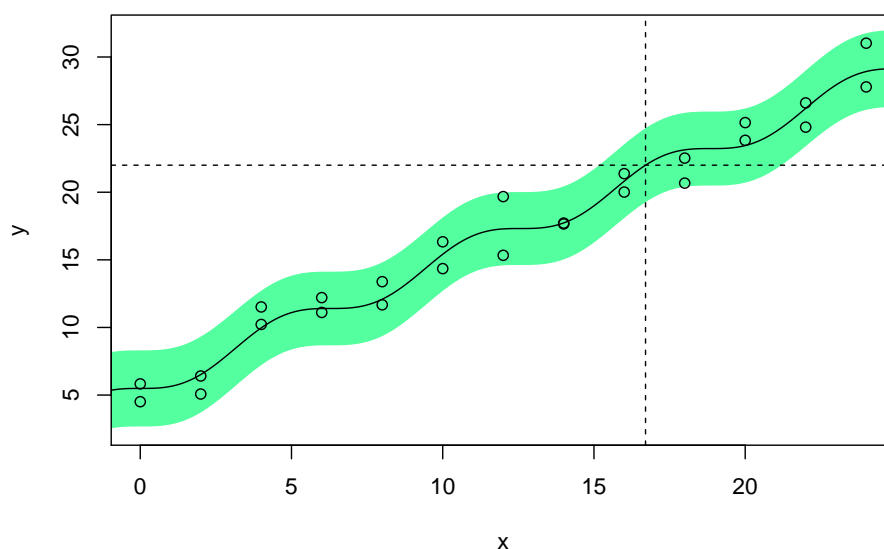


Figure 4: Scatterplot of simulated data with fitted calibration line and pointwise prediction band. A horizontal reference line is drawn through the observed value $y_0 = 22$. The vertical line identifies the position of the point estimate \hat{x}_0 .

```

}

## Variance-covariance matrix of (theta1.hat, theta2.hat, Y0)'
covmat <- diag(3)
covmat[1:2, 1:2] <- vcov(mod)
covmat[3, 3] <- summary(mod)$sigma^2

## Numerically evaluate gradient
params <- c(coef(mod), y0 = 22)
grad <- attr(numericDeriv(quote(x0.fun(params)), "params"), "gradient")

## Calculate standard error
(se <- sqrt(grad %*% covmat %*% t(grad)))

```

The following example uses the `nasturtium` data from the `drc` package. These data were analyzed in [Racine-Poon \(1988\)](#) using an approximate Bayesian approach. Bioassays were performed on a type of garden cress called nasturtium. Six replicates of the response (plant weight in mg after the third week of growth) were measured at seven preselected concentrations of an agrochemical. The weights corresponding to three new soil samples, all sharing the same (unknown) concentration x_0 , were observed to be 309, 296, and 419 mg. The block of code below fits a log-logistic model

$$f(x; \theta) = \begin{cases} \theta_1, & x = 0, \\ \theta_1 / [1 + \exp\{\theta_2 + \theta_3 \ln(x)\}], & x > 0, \end{cases}$$

and computes an approximate 95% inversion interval for x_0 .

```

## Load package containing nasturtium data
library(drc)

## Fit log-logistic model
mod <- nls(weight ~ theta1/(1 + exp(theta2 + theta3 * log(conc))),
  start = list(theta1 = 1000, theta2 = -1, theta3 = 1),
  data = nasturtium)
plotFit(mod, interval = "prediction") # figure not shown

## Compute approximate 95% inversion interval
invest(mod, y0 = c(309, 296, 419), interval = "inversion")

```

The interval obtained is (1.7722, 2.9694) with a point estimate of 2.2639. We can check the point

estimate manually. Some algebra gives

$$\hat{x}_0 = \exp \left\{ \frac{\log(\hat{\theta}_1 / \bar{y}_0 - 1) - \hat{\theta}_2}{\hat{\theta}_3} \right\} = 2.2639.$$

A Wald interval can also be obtained by running the following line of code:

```
invest(mod, y0 = c(309, 296, 419), interval = "Wald")
```

the output for which is

```
estimate    lower    upper    se
2.2639     1.6889     2.8388    0.2847
```

In certain cases (i.e., when \hat{x}_0 can be written in closed-form), one can use the very useful `deltaMethod` function from the `car` package (Fox and Weisberg, 2011) to compute the approximate standard error used in the Wald interval. For the nasturtium example, the minimal code to obtain $\widehat{se}(\hat{x}_0)$ is

```
## Using the deltaMethod function in the car package
library(car)
covmat <- diag(4)
covmat[1:3, 1:3] <- vcov(mod)
covmat[4, 4] <- summary(mod)$sigma^2 / 3 # since length(y0) = 3
(se <- deltaMethod(c(coef(mod), y0.bar = mean(c(309, 296, 419))),
  g = "exp((log(theta1 / y0.bar - 1) - theta2) / theta3)",
  vcov. = covmat)$SE)
```

which produces an estimated standard error of 0.2847019, the same as that obtained automatically by `invest`.

As indicated by the bootstrap distribution obtained in the next section, the symmetric Wald interval obtained here seems unrealistic for this problem. In the next section, we show how to obtain a *bias-corrected and accelerated* (BC_a) bootstrap confidence interval for x_0 using the `boot` package (Canty and Ripley, 2014).

Bootstrap calibration intervals

The bootstrap (Efron, 1979) provides an alternative means for computing calibration intervals. This is useful in nonlinear calibration problems where inference traditionally relies on large samples, approximate normality, and linear approximations. A practical guide to the bootstrap is provided by Davison and Hinkley (1997) and the companion R package `boot`, and bootstrap resampling for controlled calibration is discussed in Jones and Rocke (1999). Although it is likely for a "bootstrap" option to appear in a future release of `investr`, it is quite simple to set up using the recommended `boot` package. First, we discuss a naive approach to calculating bootstrap calibration intervals.

A simple, but ultimately wrong approach to resampling in controlled calibration is demonstrated in the following example for the nasturtium data:

```
library(boot)

## Function to compute estimate of x0
x0.fun <- function(object, y) {
  theta <- unname(coef(object))
  exp((log(theta[1] / mean(y) - 1) - theta[2]) / theta[3])
}

## Bootstrap setup
y0 <- c(309, 296, 419)
res <- resid(mod) - mean(resid(mod)) # center the residuals
n <- length(res)
boot.data <- data.frame(nasturtium, res = res, fit = fitted(mod))
boot.fun <- function(data, i) {
  boot.mod <- nls(fit + res[i] ~ theta1 / (1 + exp(theta2 + theta3 * log(conc))),
    start = list(theta1 = 1000, theta2 = -1, theta3 = 1), data = data)

  ## Make sure the original estimate also gets returned
```

```

    if (all(i == 1:n)) x0.fun(mod, y0) else x0.fun(boot.mod, y0)
  }

## Run bootstrap simulation (takes about 50s on a standard laptop)
set.seed(123) # for reproducibility
res <- boot(boot.data, boot.fun, R = 9999) # collect 9,999 bootstrap samples
boot.ci(res, type = "bca") # obtain BCa confidence interval for x0

```

This produces an approximate 95% BC_a calibration interval of (2.04, 2.52), quite shorter than the ones produced by the inversion and Wald methods in the previous section. Did the bootstrap really do that well? The answer here is no, but it is not the bootstrap's fault. Recall that \bar{y}_0 is an observed value of the random variable \bar{Y}_0 which has variance σ^2/m . This source of variability is ignored in the bootstrap Monte Carlo simulation above, which treats \bar{y}_0 as a fixed constant. Compare the interval just obtained with the output from

```
invest(mod, y0 = mean(c(309, 296, 419)), mean.response = TRUE)
```

We can get a hold of this extra source of variability by again sampling with replacement from the centered residuals. In particular, we resample the responses y_{0j} ($j = 1, 2, \dots, m$), by randomly selecting m additional residuals e_j^* from the centered residuals and calculating the bootstrap responses

$$y_{0j}^* = \bar{y}_0 + e_j^*, \quad j = 1, 2, \dots, m.$$

Let $\bar{y}_0^* = \sum_{j=1}^m y_{0j}^*/m$. The correct bootstrap replicate of x_0 is given by $\hat{x}_0^* = f^{-1}(\bar{y}_0^*; \hat{\theta}^*)$. To implement this, all we need to do is make the following changes to `boot.fun` (the changes are colored **magenta**):

```

boot.fun <- function(data, i) {
  boot.mod <- nls(fit + res[i] ~ theta1 / (1 + exp(theta2 + theta3 * log(conc))),
    start = list(theta1 = 1000, theta2 = -1, theta3 = 1), data = data)

  ## Simulate the correct variance
  Y0 <- y0 + sample(data$res, size = length(y0), replace = TRUE)

  ## Make sure the original estimate also gets returned
  if (all(i == 1:n)) x0.fun(mod, y0) else x0.fun(boot.mod, Y0)
}

```

Rerunning the previous example with the modified `boot.fun` function produces a more realistic 95% confidence interval for x_0 of (1.818, 2.950) and an estimated standard error for \hat{x}_0 of 0.2861 (similar to that obtained by the delta method earlier). Figure 5 shows a histogram and normal Q-Q plot of the 9,999 bootstrap replicates of \hat{x}_0 . The bootstrap distribution of \hat{x}_0 is comparable to the approximate posterior density of x_0 given in [Racine-Poon \(1988, Fig. 9\)](#). The bootstrap distribution is skewed to the right and clearly not normal, suggesting that the Wald interval may not be realistic for this problem.

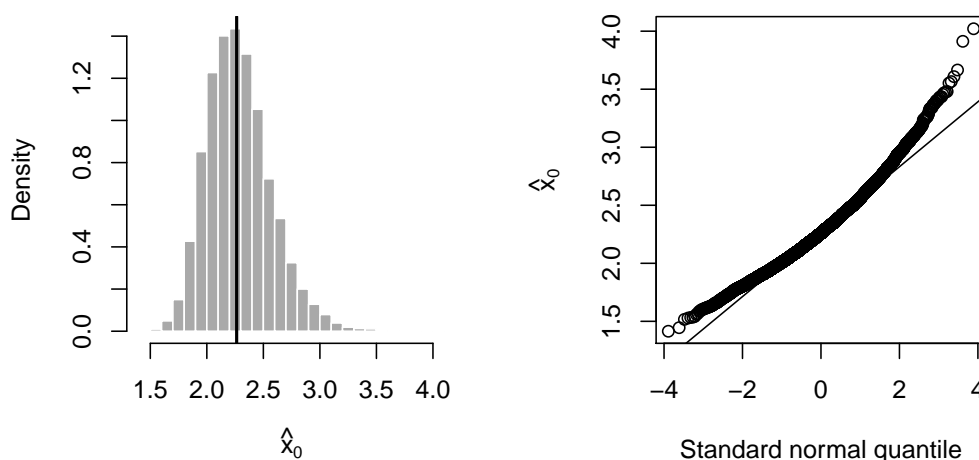


Figure 5: The left panel shows a histogram of the bootstrap replicates with a vertical line indicating the position of the original estimate \hat{x}_0 . The right panel shows a normal Q-Q plot of the bootstrap replicates.

Summary

We introduced the **investr** package for computing point estimates along with inversion and Wald-based confidence intervals for linear and nonlinear calibration problems with constant variance. We also showed how the **boot** package can be used for constructing approximate $100(1 - \alpha)\%$ calibration intervals, which for convenience, may be incorporated into a future release of **investr**. The authors are currently working on extending the package to handle the case of heteroscedastic errors, random coefficient models (e.g., objects of class "lme" from the recommended **nlme** package; Pinheiro et al., 2014), and even multivariate calibration problems (e.g., objects of class "mlm").

Acknowledgements

The authors would like to thank two anonymous reviewers and the Editor for their helpful comments and suggestions.

Bibliography

- D. M. Bates and D. G. Watts. *Nonlinear Regression Analysis and its Applications*. Wiley Series in Probability and Statistics: Probability and Statistics Section. John Wiley & Sons, New York, 1988. [p90, 94]
- P. Brown. *Measurement, Regression, and Calibration*. Oxford Science Publications. Clarendon Press, 1993. [p94]
- A. Canty and B. Ripley. *boot: Bootstrap R (S-Plus) Functions*, 2014. URL <http://CRAN.R-project.org/package=boot>. R package version 1.3-11. [p97]
- A. C. Davison and D. V. Hinkley. *Bootstrap Methods and their Applications*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, Cambridge, 1997. [p97]
- R. Dorfman. A note on the δ -method for finding variance formulae. *The Biometric Bulletin*, 1:129–137, 1938. [p92]
- N. R. Draper and H. Smith. *Applied Regression Analysis*. Wiley Series in Probability and Mathematical Statistics: Applied Probability and Statistics. John Wiley & Sons, New York, 1998. [p91]
- B. Efron. Bootstrap methods: Another look at the jackknife. *Annals of Statistics*, 7(1):1–26, 1979. [p97]
- C. Eisenhart. The interpretation of certain regression methods and their use in biological and industrial research. *Annals of Mathematical Statistics*, 10(2):162–186, 1939. [p90]
- E. C. Fieller. Some problems in interval estimation. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 16:175–185, 1954. [p91]
- J. Fox and S. Weisberg. *An R Companion to Applied Regression*. Sage, Thousand Oaks CA, second edition, 2011. URL <http://socserv.socsci.mcmaster.ca/jfox/Books/Companion>. [p97]
- F. A. Graybill. *Theory and Application of the Linear Model*. Duxbury Classic Series. Duxbury, Pacific Grove, CA, 1976. [p90]
- F. A. Graybill and H. K. Iyer. *Regression Analysis: Concepts and Applications*. Duxbury Press, Belmont, CA, 1994. [p91, 93]
- B. M. Greenwell. *investr: Functions for Inverse Estimation with Linear and Nonlinear Regression Models in R*, 2014. URL <https://github.com/w108bmg/investr>. R package version 1.0.2. [p90]
- S. Huet. *Statistical Tools for Nonlinear Regression: A Practical Guide with S-PLUS and R Examples*. Springer Series in Statistics. Springer, 2004. [p94]
- G. Jones and D. M. Rocke. Bootstrapping in controlled calibration experiments. *Technometrics*, 41(3): 224–233, 1999. [p97]
- R. Miller. *Simultaneous Statistical Inference*. Springer Series in Statistics. Springer-Verlag, 1981. [p94]
- C. Osborne. Calibration: A review. *International Statistical Review*, 59(3):309–336, 1991. [p90]
- J. Pinheiro, D. Bates, S. DebRoy, D. Sarkar, and R Core Team. *nlme: Linear and Nonlinear Mixed Effects Models*, 2014. URL <http://CRAN.R-project.org/package=nlme>. R package version 3.1-117. [p99]

- A. Racine-Poon. A Bayesian approach to nonlinear calibration problems. *Journal of the American Statistical Association*, 83(403):650–656, 1988. [p96, 98]
- C. Ritz and J. C. Streibig. Bioassay analysis using R. *Journal of Statistical Software*, 12(5):1–22, 2005. URL <http://www.jstatsoft.org/v12/i05/>. [p94]
- J. R. Schwenke and G. A. Milliken. On the calibration problem extended to nonlinear models. *Biometrics*, 47(2):563–574, 1991. [p94]
- G. Seber and A. Lee. *Linear Regression Analysis*. Wiley Series in Probability and Statistics. John Wiley & Sons, 2003. [p94]
- G. Seber and C. Wild. *Nonlinear Regression*. Wiley Series in Probability and Statistics. John Wiley & Sons, 2003. [p94, 95]
- 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>. [p94]

Brandon M. Greenwell
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433
United States of America
brandon.greenwell@afit.edu

Christine M. Schubert Kabban
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433
United States of America
christine.schubertkabban@afit.edu

Rankcluster: An R Package for Clustering Multivariate Partial Rankings

by Julien Jacques, Quentin Grimonprez and Christophe Biernacki

Abstract The **Rankcluster** package is the first R package proposing both modeling and clustering tools for ranking data, potentially multivariate and partial. Ranking data are modeled by the Insertion Sorting Rank (ISR) model, which is a meaningful model parametrized by a central ranking and a dispersion parameter. A conditional independence assumption allows multivariate rankings to be taken into account, and clustering is performed by means of mixtures of multivariate ISR models. The parameters of the cluster (central rankings and dispersion parameters) help the practitioners to interpret the clustering. Moreover, the **Rankcluster** package provides an estimate of the missing ranking positions when rankings are partial. After an overview of the mixture of multivariate ISR models, the **Rankcluster** package is described and its use is illustrated through the analysis of two real datasets.

Introduction

Ranking data occur when a number of observers are asked to rank a list of objects according to their personal preference order. Such data are of great interest in human activities involving preferences, attitudes or choices like Psychology, Sociology, Politics, Marketing, *etc.* For instance, the voting system *single transferable vote*, used in Ireland, Australia and New Zealand, is based on preferential voting (Gormley and Murphy, 2008). In many applications, the study of ranking data discloses heterogeneity, due for instance to different political meanings, different human preferences, *etc.* Clustering analysis is then a useful tool for statistical exploration of such datasets, by characterizing the datasets in terms of a reduced number of clusters within which rankings are similar. In the present work, the clusters will be characterized by a median ranking (location parameter) and by a dispersion parameter. The interpretation of these two parameters for each cluster will produce a comprehensive and synthetic analysis of the whole dataset.

In the present paper we illustrate these ideas with two studies in sociology and in sport. In the sociology study, several students were asked to rank five words according to strength of association (least to most associated) with the target word "Idea": Thought, Play, Theory, Dream and Attention. The cluster analysis of the resulting rankings from this study allows the population of students to be characterized in terms of three clusters that can be interpreted (thanks to the parameters specific to each cluster) as follows: scientific students, literary-minded students and a third group of students in an intermediate position between scientific and too literary-minded. The second study focuses on the performance of English football clubs during the last two decades, and more particularly those of the "Big Four" clubs: Manchester United, Liverpool, Arsenal and Chelsea. For this, the rankings of these four clubs at the English championship (Premier League) and according to the UEFA coefficients are analyzed. The cluster analysis exhibits two relatively similar clusters, reflecting a certain stability in the hierarchy between these clubs, except for the position of Chelsea within the hierarchy: it appears that Chelsea obtains better results in the second half of the period 1993-2013, which can be probably explained by the acquisition of the club by a Russian investor in 2003.

Recently, Jacques and Biernacki (2012) proposed a model-based clustering algorithm in order to analyse and explore such ranking data. This algorithm is able to take into account multivariate rankings (when several rankings are simultaneously observed, as for instance in the sport application in which rankings at the Premier League and rankings according to the UEFA coefficient are observed simultaneously each year) with potential partial rankings (when an observer did not rank all the objects). To the best of our knowledge, this is the only clustering algorithm for ranking data with such a wide application scope. This algorithm is based on an extension of the Insertion Sorting Rank (ISR) model (Biernacki and Jacques, 2013) for ranking data, which is a meaningful and effective model obtained by assuming that the ranking generating process is a sorting algorithm. The ISR model is parametrized by a location parameter, the *modal ranking*, and a dispersion parameter. The heterogeneity of the rank population is modeled by a mixture of ISR models and a conditional independence assumption allows the extension to multivariate rankings. Maximum likelihood estimation is performed through a SEM-Gibbs algorithm, in which partial rankings are considered as missing data, which allows them to be simulated during the estimation process.

This algorithm has been implemented in C++ and is available through the **Rankcluster** package for R, available on the CRAN website and presented in depth in the sequel of this paper.

The paper is organized as follows: the next section briefly presents the clustering algorithm

proposed in [Jacques and Biernacki \(2012\)](#). Then, the following sections describe the existing R packages dedicated to ranking data, and the functionality of the **Rankcluster** package. The **Rankcluster** package is then illustrated through the cluster analysis of two datasets: 1. The *words* dataset of [Fligner and Verducci \(1986\)](#) featuring univariate rankings without any missing ranking positions, and 2. the rankings of the Big Four English Football clubs (Manchester United, Liverpool, Arsenal and Chelsea) according to the Premier League results and to their UEFA coefficients between 1993 and 2013, featuring bivariate rankings with some missing ranking positions.

Overview of the model-based clustering algorithm

This section gives an overview of the model-based clustering algorithm for multivariate partial rankings proposed in [Jacques and Biernacki \(2012\)](#). It relies on the univariate ISR model that we introduce first.

The univariate ISR model

Rank data arise when judges or observers are asked to rank several objects $\mathcal{O}_1, \dots, \mathcal{O}_m$ according to a given order of preference. The resulting ranking can be designated by its *ordering* representation $x = (x^1, \dots, x^m) \in \mathcal{P}_m$ which signifies that Object \mathcal{O}_{x^h} is the h th ($h = 1, \dots, m$), where \mathcal{P}_m is the set of the permutations of the first m integers, or by its *ranking* representation $x^{-1} = (x_1^{-1}, \dots, x_m^{-1})$, which contains the ranks assigned to the objects and means that \mathcal{O}_i is in x_i^{-1} th position ($i = 1, \dots, m$). In the **Rankcluster** package, the ranking representation is used, but users working with the ordering representation can use a function implemented in the package in order to convert their data from one representation to the other.

Based on the assumption that a rank datum is the result of a sorting algorithm based on paired comparisons, and that the judge who ranks the objects uses insertion sort because of its optimality properties (minimum number of paired comparisons), [Biernacki and Jacques \(2013\)](#) stated the following so-called ISR model:

$$p(x; \mu, \pi) = \frac{1}{m!} \sum_{y \in \mathcal{P}_m} p(x|y; \mu, \pi) = \frac{1}{m!} \sum_{y \in \mathcal{P}_m} \pi^{G(x,y,\mu)} (1 - \pi)^{A(x,y) - G(x,y,\mu)},$$

where

- $\mu \in \mathcal{P}_m$, the modal ranking, is a *location parameter*. Its *opposite* ranking $\bar{\mu}$ ($\bar{\mu} = \mu \circ \bar{e}$ with $\bar{e} = (m, \dots, 1)$) is the rank of smallest probability,
- $\pi \in [\frac{1}{2}, 1]$ is the probability of good paired comparison in the sorting algorithm (where *good* means in accordance with the modal ranking μ). It is a *scale parameter*: the distribution is uniform when $\pi = \frac{1}{2}$ and the mode μ of the distribution is uniformly more pronounced as π grows, being a Dirac function in μ when $\pi = 1$,
- the sum over $y \in \mathcal{P}_m$ corresponds to all the possible initial presentation orders of the objects to be ranked (with identical prior probabilities equal to $1/m!$ because they are unknown),
- $G(x, y, \mu)$ is the number of good paired comparisons during the sorting process leading to return x when the presentation order is y ,
- $A(x, y)$ corresponds to the total number of paired comparisons (good or wrong).

The precise definitions of $G(x, y, \mu)$ and $A(x, y)$ can be found in [Biernacki and Jacques \(2013\)](#).

Mixture of multivariate ISR

Now redefine $\mathbf{x} = (x^1, \dots, x^J) \in \mathcal{P}_{m_1} \times \dots \times \mathcal{P}_{m_J}$ as a *multivariate rank*, in which $x^j = (x^{j1}, \dots, x^{jm_j})$ is a rank of m_j objects ($1 \leq j \leq J$).

The population of multivariate ranks is assumed to be composed of K groups in proportions p_k ($p_k \in [0, 1]$ and $\sum_{k=1}^K p_k = 1$). Given a group k , the J components x^1, \dots, x^J of the multivariate rank datum \mathbf{x} are assumed to be sampled from independent ISR distributions with corresponding modal rankings μ_k^1, \dots, μ_k^J (each $\mu_k^j \in \mathcal{P}_{m_j}$) and good paired comparison probabilities $\pi_k^1, \dots, \pi_k^J \in [\frac{1}{2}, 1]$.

The unconditional probability of a rank \mathbf{x} is then

$$p(\mathbf{x}; \theta) = \sum_{k=1}^K p_k \prod_{j=1}^J p(x^j; \mu_k^j, \pi_k^j),$$

where $\theta = (\pi_k^j, \mu_k^j, p_k)_{k=1, \dots, K, j=1, \dots, J}$ and $p(x^j; \mu_k^j, \pi_k^j)$ is defined by (22.2.1).

Each component x^j of \mathbf{x} can be full or partial. Frequently, the objects in the top positions will be ranked and the missing ones will be at the end of the ranking, but our model does not impose this and is able to work with partial rankings whatever the positions of the missing data (see details in [Jacques and Biernacki \(2012\)](#)).

Estimation algorithm

Parameter estimation is performed by maximum likelihood. However, maximum likelihood estimation is not straightforward since the model contains some missing data: the cluster memberships z_i of the observations, the presentation orders \mathbf{y}_i (where $\mathbf{y}_i = (y_i^1, \dots, y_i^J)$ is the presentation order for the i th observation) and the unobserved ranking positions, denoted by $\hat{\mathbf{x}}_i$ (for partial rankings). In such a situation, a convenient way to maximize the likelihood is to consider an EM algorithm ([Dempster et al., 1977](#)). This algorithm relies on the completed-data log-likelihood, and proceeds by iterating an estimation (E) step, in which the conditional expectation of the completed-data log-likelihood is computed, and a maximization (M) step, in which the model parameters are estimated by maximizing the conditional expectation computed in the E step. Unfortunately, the EM algorithm is tractable only for univariate full rankings with moderate m ($m \leq 7$) for mathematical and numerical reasons. In particular, when partial rankings occur, the E step is intractable since the completed-data log-likelihood is not linear for all three types of missing data (refer to [Jacques and Biernacki \(2012\)](#) for its expression). A Stochastic EM (SEM) Gibbs approach is then proposed in [Jacques and Biernacki \(2012\)](#) to overcome these problems.

The fundamental idea of the SEM-Gibbs algorithm is to reduce the computational complexity that is present in both E and M steps of EM by removing all explicit and extensive use of the conditional expectations of any product of missing data. First, it relies on the SEM algorithm ([Geman and Geman, 1984](#); [Celeux and Diebolt, 1985](#)) which generates the latent variables at a so-called stochastic (S) step from the conditional probabilities computed at the E step. Then these latent variables are directly used in the M step. Second, the advantage of the SEM-Gibbs algorithm in comparison with the basic SEM one is that the latent variables are generated without calculating conditional probabilities at the E step, thanks to a Gibbs algorithm. Refer to [Jacques and Biernacki \(2012\)](#) for more details.

Existing R packages for ranking data

To the best of our knowledge, there are only two packages dedicated to the analysis of ranking data, available on the CRAN website, but their functionalities are significantly limited in comparison to our package **Rankcluster**, as we discuss now:

- The **pmr** package ([Lee and Yu, 2013](#)) provides some descriptive statistics and modeling tools using classical rank data models for full univariate ranking data: Luce models, distance-based models, and rank-ordered logit (see [Marden \(1995\)](#) for a description of these models). Visualization of ranking data using polytopes is also available for less than four objects to rank ($m \leq 4$).
- The **RMallow** package ([Gregory, 2012](#)) performs clustering of univariate ranking data using mixtures of distance-based models ([Murphy and Martin, 2003](#)).

Rankcluster proposes modeling and clustering tools on the basis of the mixture of multivariate ISR presented in the previous section. Comparing to the existing packages, **Rankcluster** is the only package taking into account multivariate and partial ranking data.

Overview of the Rankcluster functions

This section presents briefly the functions of the **Rankcluster** package. For more details, refer to the help of the functions.

The main function: `rankclust()`

The `rankclust()` function performs modeling or cluster analysis of a dataset of rankings. For cluster analysis, the number of clusters can be specified by the user, or selected by the Bayesian Information Criterion (BIC) ([Schwarz, 1978](#)), or the Integrated Completed Likelihood (ICL) ([Biernacki et al., 2000](#)) in a list of possible numbers provided by the user. By default, the number of clusters is equal to one,

and consequently a single homogeneous multivariate ISR model is fitted to the data. The main inputs and outputs are described below.

This function has only one mandatory argument, `data`, which is a matrix composed of the n observed ranks in their ranking representation. For **univariate rankings** the number of columns of data is m (default value of argument `m`). For **multivariate rankings**, data has $m_1 + \dots + m_J$ columns: the first m_1 columns contain x^1 (first dimension), the columns $m_1 + 1$ to $m_1 + m_2$ contain x^2 (second dimension), and so on. In this case, the argument `m` must be filled with the vector of size (m_1, \dots, m_J) . Several arguments allow the different tuning parameters used in the SEM-Gibbs estimation to be specified. Refer to [Jacques and Biernacki \(2012\)](#) and to `help(rankclust)` for more details (section 22.5 also gives some examples). The option `run` allows the number of initializations of the SEM-Gibbs algorithm to be set (1 by default). In the case of multiple initializations, the best solution according to the approximated log-likelihood is retained.

The `rankclust()` function returns an instance of the "ResultTab" class. It contains 5 slots, including results which is a list containing all $k = 1, \dots, K$ classes, summarized in 18 slots, among which the main ones are:

- `proportion`: a K -vector of proportions p_1, \dots, p_K .
- `pi`: a $K \times J$ -matrix composed of the scale parameters π_k^j ($1 \leq k \leq K$ and $1 \leq j \leq J$).
- `mu`: a matrix with K lines and $m_1 + \dots + m_J$ columns in which line k is composed of the location parameters $(\mu_k^1, \dots, \mu_k^J)$ of cluster k .
- `ll, bic, icl`: values of the log-likelihood, BIC and ICL, respectively.
- `tik`: an $n \times K$ matrix containing the estimates of the conditional probabilities of belonging to each cluster for the observed ranks.
- `partition`: an n -vector containing the partition estimate resulting from the clustering.
- `partialRank`: a matrix containing the full rankings, estimated using the within cluster ISR parameters when the ranking is partial.
- `distanceProp, distancePi, distanceMu`: distances between the final estimate and the current value at each iteration of the SEM-Gibbs algorithm (except the burn-in phase) for respectively: proportions p_k , scale parameters π_k^j , location parameters μ_k^j .
- `distanceZ`: a vector of size $Q_{\text{sem-Bsem}}$ containing the Rand index ([Rand, 1971](#)) between the final estimated partition and the current value at each iteration of the SEM-Gibbs algorithm (except the burning phase).

If `res` is the name assigned to the result of `rankclust()`, each slot can be obtained by `res[k]@slotname`, where k is the cluster index and `slotname` is the name of the selected slot (`proportion, pi ...`). For the slots `ll, bic` and `icl`, `res["slotname"]` returns a vector of size K containing the values of the slot for each cluster index.

Companion functions

In addition to the main function, `rankclust()`, several companion functions are available in **Rankcluster**:

- `convertRank()`: converts ranking representation x^{-1} of a rank to its ordering representation x , and *vice-versa* since $x \circ x^{-1} = x^{-1} \circ x$.
- `distCayley()`, `distHamming()`, `distKendall()`, `distSpearman()`: compute usual distances between rankings ([Marden, 1995](#)) for either ranking or ordering representation.
- `frequence()`: transforms a raw dataset composed of a matrix of ranks (one rank per line, with possibly equal lines if the same rank is observed several times) into a matrix rank/frequency containing in line each different observed ranks and one additional last column with the frequency of observation of these ranks. Conversely, `unfrequence()` transforms a rank/frequency dataset in a raw dataset, as requested in input argument of `rankclust()`.
- `khi2()`: performs a chi-squared goodness-of-fit tests and returns the p-value of the test (See [Biernacki and Jacques \(2013\)](#) for details).
- `kullback()`: estimates the Kullback-Leibler divergence between two ISR models.
- `simulISR()`: simulates a univariate and unimodal dataset of full rankings according to the ISR model.
- `probability()`: computes the probability of a ranking (multivariate or not) according to the ISR model.

Rankcluster through examples

This section illustrates the use of the `rankclust()` function on two real datasets. The first one, `words`, is a well-known dataset in ranking studies, due to [Fligner and Verducci \(1986\)](#), which consists of words associations by students. The second one, `big4` consists of the rankings of the Big Four of English Football (Manchester United, Liverpool, Arsenal and Chelsea) according to the Premier League results and to their UEFA coefficients between 1993 and 2013.

The words dataset

This dataset was collected under the auspices of the Graduate Record Examination Board ([Fligner and Verducci, 1986](#)). A sample of 98 college students were asked to rank five words according to strength of association (least to most associated) with the target word "Idea": 1 = Thought, 2 = Play, 3 = Theory, 4 = Dream and 5 = Attention.

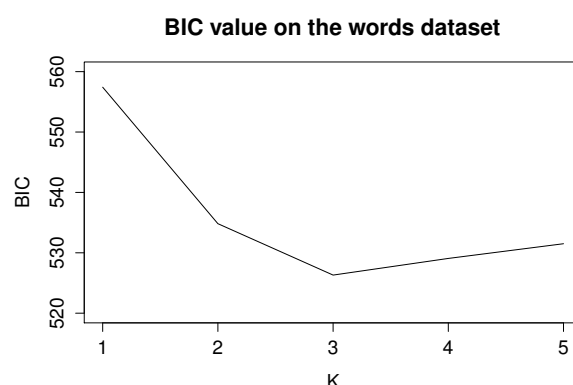


Figure 1: Value of the BIC criterion with mixture of ISR for the words dataset.

First we start by installing and loading the **Rankcluster** package and then loading the `words` dataset:

```
R> install.packages("Rankcluster")
R> library(Rankcluster)
R> data(words)
```

Using the `rankclust()` function, a clustering with respectively 1 to 5 clusters is estimated:

```
R> res=rankclust(words$data, m=words$m, K=1:5, Qsem=1000, Bsem=100, Ql=500, Bl=50,
+ maxTry=20, run=10)
```

The number of SEM-Gibbs iterations (Q_{sem}) has been set to 1000, with a burn-in phase of 100 iterations (B_{sem}). For likelihood approximation the numbers of iterations (Q_l and B_l) have been divided by two. Option `maxTry=20` allows the estimation algorithm to be restarted up to 20 times if one cluster becomes empty (which frequently occurs for $K = 5$). Finally, the SEM-Gibbs algorithm is initialized 5 times (`run=5`), and the best solution (according to the approximated likelihood) is retained. Computing time on a laptop with 2.80GHz CPU is about 3 minutes (7 seconds per run per number of clusters). The reader who wants to test more quickly the package can reduced the number of runs, using for instance:

```
R> res=rankclust(words$data, m=words$m, K=1:5, Qsem=200, Bsem=20, Ql=200, Bl=20,
+ maxTry=20)
```

The values of the BIC criterion, obtained by `res["bic"]` and plotted on Figure 1, tend to select three clusters.

The parameter estimates for $K = 3$ are given below for proportions p_k , scales π_k and modes μ_k :

```
R> res[3]@proportion
[1] 0.3061224 0.4918367 0.2020408
R> res[3]@pi
dim 1
```

```

c1 1 0.9060649
c1 2 0.9416822
c1 3 0.8642753
R> res[3]@mu
      dim 1
c1 1      5 1 3 4 2
c1 2      5 1 4 3 2
c1 3      5 2 4 3 1

```

The word Thought is most associated with Idea for all clusters. Looking at the rankings of the four other words can suggest an interesting interpretation of the clusters. Indeed, the first cluster, composed of about 30% of the students, is characterized by the following modal ranking: Play, Attention, Theory, Dream, Thought. Students in this cluster are probably literary-minded students who rank the word Dream just after Thought. Students in the second cluster (about half of total students) are probably more scientific since they rank the word Theory just after Thought, and so before the word Dream: Play, Attention, Dream, Theory, Thought. This cluster is also the most homogeneous, with a high scale parameter value (low dispersion): $\pi_2 \simeq 0.94$. Finally, the last cluster is characterized by the following mode: Attention, Play, Dream, Theory, Thought. The only difference in the modal ranking with the scientific students is the preference of Play rather than Attention. This cluster, which is the smallest (20% of the students), can be qualified as intermediate cluster, probably composed of a set of students not too scientific or too literary-minded, as evidenced by the smallest of the three scale parameter values ($\pi_3 \simeq 0.86$).

The big4 dataset

In the two last decades, English football has been dominated by four clubs, forming the “Big Four”: Manchester United, Liverpool, Arsenal and Chelsea. In this example, we analyse the rankings of these four teams in the English championship (Premier League) and their rankings according to the UEFA coefficient, a European statistic on football teams based on the results of European football competitions and used for ranking and seeding teams in international competitions. The big4 dataset, available in the package, is composed of Premier League rankings and UEFA rankings from 1993 to 2013, in ranking notation (club “1” is Manchester United, “2” is Liverpool, “3” is Arsenal and “4” is Chelsea). In 2001 Arsenal and Chelsea had the same UEFA coefficient and then are tied for the first ranking dimension. With **Rankcluster**, one way to take into account such ties in ranking data is to consider the corresponding ranking positions as missing: the UEFA ranking becomes then (1, 0, 0, 2) for 2001, what means that Manchester United is the first, Liverpool is the last, and the two intermediate positions are for Arsenal and Chelsea in an unknown order.

First, the big4 dataset is loaded:

```
R> data(big4)
```

Then, the number of clusters is estimated with the BIC criterion. For this, clustering for 1 to 3 clusters is performed with the `rankclust()` function. It should be noted that, for 3 clusters, the algorithm has to be launched several times since it often converges to a solution with one empty cluster. The values of the BIC criterion are plotted on Figure 2, and tend to select two groups (which confirms that with 3 clusters the estimation algorithm often converges to one empty cluster).

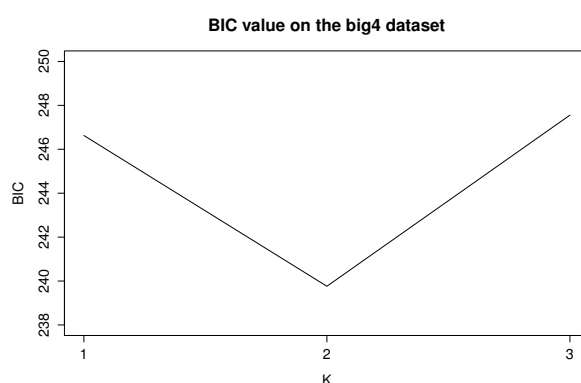


Figure 2: Values of the BIC criterion with mixture of ISR for the big4 dataset.

The clustering with $K = 2$ is obtained in about 25 seconds on a laptop 2.80GHz CPU:

```
R> res = rankclust(big4$data, m=big4$m, K=2, Qsem=1000, Bsem=100, Ql=500, Bl=50,
+ maxTry=20, run=5)
```

The printed outputs for $K = 2$ are given below: value of the log-likelihood (11), values of BIC and ICL criteria, estimate of the proportions p_k 's, the modes μ_k^j 's, the scales π_k^j 's, the estimated partition and finally the conditional probability that the observations belong to each cluster (t_{ik}).

```
R> res[2]
*****
Number of clusters: 2
*****
ll= -108.5782
bic = 244.5571
icl = 253.5601
proportion: 0.3854034 0.6145966
mu:
      dim1      dim2
c11  1 3 2 4    1 3 2 4
c12  1 3 4 2    1 4 3 2
pi:
      dim1      dim2
c11 0.9698771 0.7759781
c12 0.6945405 0.7707101
partition:
[1] 2 1 1 1 2 1 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2
tik:
[,1] [,2]
[1,] 6.280426e-01 0.371957427
[2,] 9.933149e-01 0.006685094
[3,] 9.565052e-01 0.043494814
[4,] 9.923445e-01 0.007655450
[5,] 6.609592e-01 0.339040841
[6,] 9.916482e-01 0.008351815
[7,] 1.215925e-03 0.998784075
[8,] 3.500066e-01 0.649993363
[9,] 3.441535e-02 0.965584647
[10,] 4.063712e-01 0.593628767
[11,] 9.589745e-01 0.041025518
[12,] 9.751644e-01 0.024835551
[13,] 6.806917e-02 0.931930833
[14,] 3.175113e-03 0.996824887
[15,] 1.263591e-03 0.998736409
[16,] 7.713598e-06 0.999992286
[17,] 9.115424e-04 0.999088458
[18,] 1.734860e-01 0.826513968
[19,] 1.605939e-04 0.999839406
[20,] 7.425989e-02 0.925740107
[21,] 2.171738e-02 0.978282624
*****
```

The estimated clustering exhibits two groups, the second one being larger than the first one ($p_1 \simeq 0.39$ and $p_2 \simeq 0.61$). The values of the modes for each cluster and dimension lead to two interesting remarks. First, the ranking in each dimension is very similar in both clusters: exactly the same for cluster 1 and just one transposition in the last two positions for cluster 2. This means that the performance of the clubs at the Premier League is highly correlated with their UEFA rankings, which is related to the results of the clubs in the European competitions over the previous five seasons. This first comment shows a certain inertia in the performance of the clubs. Secondly, the distinction between the two clusters is essentially due to the position of Chelsea (club 4). Indeed, in the first cluster Chelsea is the last in both rankings, but it is in second position in the second cluster. Moreover, in the partition, we find that cluster 2 is mainly present in the second half of the period 1993-2013 (see for instance the conditional probabilities of cluster membership on Figure 3). This rise of Chelsea can be explained by the acquisition of the club by Russian investor Roman Abramovich in 2003, who brought great players to the club.

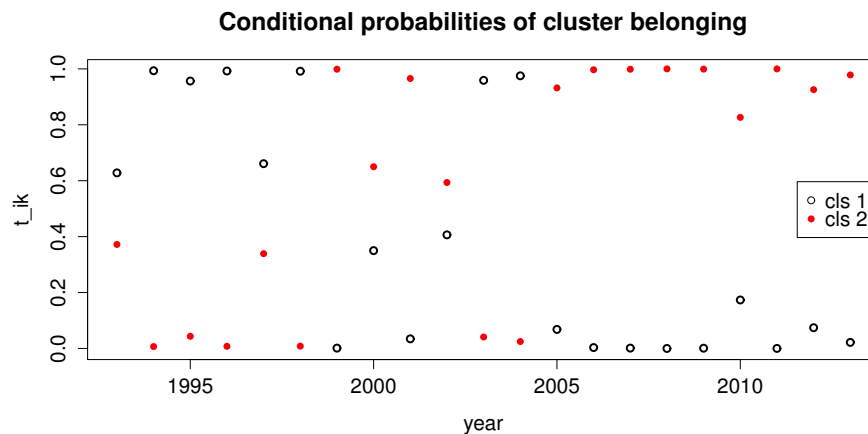


Figure 3: Conditional probabilities for each observation (year) to belong to cluster 1 (black circle) and 2 (red filled circle).

In addition to this information, the `summary()` function gives an overview of the partition by printing the five ranks of highest probability and the five ranks of highest entropy for each cluster:

```
R> summary(res)
```

The ranks of highest probability are the best representatives of the cluster, whereas the ranks of highest entropy are those for which their membership to the cluster are the less obvious. Note that the full list of the cluster members with their probabilities and entropies are available through the slots `probability` and `entropy`. Table 1 gives an example of these outputs for cluster 2. The rankings of 2011 are the most representative of the cluster, and the five most representative rankings of the cluster correspond to rankings after 2003. Similarly, the four observations whose membership in cluster 2 is the most questionable correspond to observations before 2003. This confirms the previous analysis indicating that cluster 2 is due to the rise of Chelsea.

year	UEFA	Prem. League	probability
2011	(1,3,4,2)	(1,4,3,2)	2.367e-04
2008	(4,2,3,1)	(1,4,3,2)	6.862e-05
2013	(2,4,3,1)	(1,4,3,2)	3.529e-05
2009	(3,2,4,1)	(1,2,4,3)	3.097e-05
2005	(1,2,3,4)	(3,4,2,1)	2.151e-05
year	UEFA	Prem. League	entropy
2002	(1,4,2,3)	(3,2,1,4)	0.6755106
1993	(1,2,3,4)	(1,2,3,4)	0.6599892
2000	(1,4,3,2)	(1,3,2,4)	0.6474507
1997	(1,2,3,4)	(1,3,2,4)	0.6403972
2010	(1,0,0,4)	(2,4,3,1)	0.4613709

Table 1: Rankings with the highest entropies and probabilities in the second cluster.

The `summary()` function also prints the estimated full ranking for each partial ranking. For instance, in 2001 Arsenal and Chelsea had the same UEFA coefficient, and when asking our model to differentiate these two teams, Arsenal is ranked before Chelsea. This is not surprising as we already remarked that the results of Chelsea were generally among the worst of the Big Four before 2003.

Finally, the variability of estimation of the model parameters can be achieved by means of the distances between the final estimate and the current value at each step of the SEM-Gibbs algorithm (refer to Jacques and Biernacki (2012) for accurate definitions of these distances). These distances are available in the slots `distanceProp`, `distancePi`, `distanceMu` of the output `res[2]`. The standard deviation of these distances can be used for instance as an indicator of estimation variability. For instance, the standard deviation of the Kendall distance (Marden, 1995) between the final estimate

of the mode and its current value at each step of the SEM-Gibbs algorithm is for cluster 2: 0.52 for the UEFA coefficients rankings and 0.43 for Premier League rankings. Similarly, the standard deviation of the scale estimate is for cluster 2 about 0.002 for both the UEFA coefficients and the Premier League rankings. Note that these variabilities are relatively small, due to the low overlapping of the two clusters (scale coefficients are quite high). In a similar way, the slot distanceZ illustrates the convergence of the SEM-Gibbs algorithm by given the Rand index (Rand, 1971) between the final partition and the current partition at each SEM-Gibbs iteration (Figure 4).

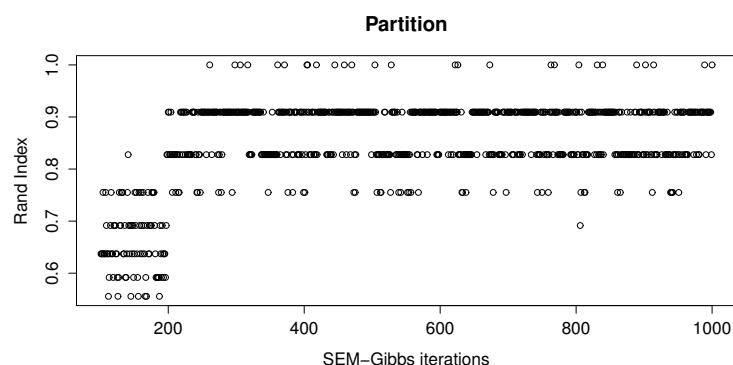


Figure 4: Evolution of the partition along with the SEM-Gibbs iterations: values of the Rand index between current and final partitions.

Conclusion

Rankcluster is the first R package dedicated to ranking data that allows modeling and cluster analysis for multivariate partial ranking data. Available on the CRAN website, this package is simple to use with its main function, `rankclust()`, having only one mandatory argument, the ranking dataset. By default a single homogeneous multivariate ISR model is fitted to the data, and specifying the number of desired clusters leads to perform a cluster analysis, with selection of the number of clusters if several numbers are given. The analysis of two real datasets presented in this paper illustrates the possibilities of the package, and also constitutes a user guide for practitioners.

Bibliography

- C. Biernacki and J. Jacques. A generative model for rank data based on insertion sort algorithm. *Computational Statistics & Data Analysis*, 58:162–176, 2013. [p101, 102, 104]
- C. Biernacki, G. Celeux, and G. Govaert. Assessing a mixture model for clustering with the integrated completed likelihood. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(4):719–725, 2000. [p103]
- G. Celeux and J. Diebolt. The SEM algorithm: A probabilistic teacher algorithm derived from the EM algorithm for the mixture problem. *Computational Statistics Quarterly*, 2(1):73–82, 1985. [p103]
- A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B. Methodological*, 39(1):1–38, 1977. With discussion. [p103]
- M. Fligner and J. Verducci. Distance based ranking models. *Journal of the Royal Statistical Society. Series B. Methodological*, 48(3):359–369, 1986. [p102, 105]
- A. Geman and D. Geman. Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Matching Intelligence*, 6:721–741, 1984. [p103]
- I. Gormley and T. Murphy. A mixture of experts model for rank data with applications in election studies. *Annals of Applied Statistics*, 2(4):1452–1477, 2008. [p101]
- E. Gregory. *RMallow: Fit Multi-Modal Mallows' Models to ranking data.*, 2012. URL <http://CRAN.R-project.org/package=RMallow>. R package version 1.0. [p103]

- J. Jacques and C. Biernacki. Model-based clustering for multivariate partial ranking data. Technical Report 8113, Inria Research Report, 2012. [p101, 102, 103, 104, 108]
- P. Lee and P. Yu. An R package for analyzing and modeling ranking data. *BMC Medical Research Methodology*, 13(65):1–11, 2013. [p103]
- J. Marden. *Analyzing and modeling rank data*, volume 64 of *Monographs on Statistics and Applied Probability*. Chapman & Hall, London, 1995. [p103, 104, 108]
- T. Murphy and D. Martin. Mixtures of distance-based models for ranking data. *Comput. Statist. Data Anal.*, 41(3-4):645–655, 2003. [p103]
- W. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 55:846–850, 1971. [p104, 109]
- G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978. [p103]

Julien Jacques
MODAL team (Inria) & Laboratoire Paul Painlevé (UMR CNRS 8524)
University Lille I
Cité Scientifique, 59655 Villeneuve d’Ascq cedex
France julien.jacques@polytech-lille.fr

Quentin Grimonprez
MODAL team
Inria Lille-Nord Europe
40 avenue du Halley, 59650 Villeneuve d’Ascq
France quentin.grimonprez@inria.fr

Christophe Biernacki
MODAL team (Inria) & Laboratoire Paul Painlevé (UMR CNRS 8524)
University Lille I
Cité Scientifique, 59655 Villeneuve d’Ascq cedex
France christophe.biernacki@math.univ-lille1.fr

The stringdist Package for Approximate String Matching

by Mark P.J. van der Loo

Abstract Comparing text strings in terms of distance functions is a common and fundamental task in many statistical text-processing applications. Thus far, string distance functionality has been somewhat scattered around R and its extension packages, leaving users with inconsistent interfaces and encoding handling. The **stringdist** package was designed to offer a low-level interface to several popular string distance algorithms which have been re-implemented in C for this purpose. The package offers distances based on counting q -grams, edit-based distances, and some lesser known heuristic distance functions. Based on this functionality, the package also offers inexact matching equivalents of R's native exact matching functions `match` and `%in%`.

Introduction

Approximate string matching is an important subtask of many data processing applications including statistical matching, text search, text classification, spell checking, and genomics. At the heart of approximate string matching lies the ability to quantify the similarity between two strings in terms of string metrics. String matching applications are commonly subdivided into online and offline applications where the latter refers to the situation where the string that is being searched for a match can be preprocessed to build a search index. In online string matching no such preprocessing takes place. A second subdivision can be made into approximate text search, where one is interested in the location of the match of one string in another string, and dictionary lookup, where one looks for the location of the closest match of a string in a lookup table. Here, we focus on dictionary lookup applications using online string matching algorithms.

String metrics may broadly be divided in edit-based distances, q -gram based distances (sometimes called n -grams) and heuristic distances. To determine edit-based distances one counts, possibly weighted, the number of fundamental operations necessary to turn one string into another. Operations may include substitution, deletion, or insertion of a character or transposition of characters. Distances based on q -grams are obtained by comparing the occurrence of q -character sequences between strings. Heuristic measures have no strong mathematical underpinning but have been developed as a practical tool with a particular application in mind. Finally, it is worth mentioning that these families of string metrics have recently been extended with metrics based on string kernels (Lodhi et al., 2002), which have been implemented in the **kernlab** package of Karatzoglou et al. (2004).

The base R installation offers functions for both string metric calculation and online text search. The `adist` function computes the generalized Levenshtein (1966) distance between strings while `agrep`, based on a library of Laurikari (2001), allows for online approximate text search based on the same distance metric. In fact, `agrep` is more advanced than simple approximate search since it allows for approximate matching against regular expressions. Furthermore, the functions `match` and `%in%` can be used for dictionary lookup based on exact matching while `pmatch` and `charmatch` can be used for lookup based on partial matches. A lookup function based on approximate string matching is not available in R's base installation.

The native R implementations are complemented by several several extension packages that offer offline string matching algorithms. For example, **RecordLinkage** (Borg and Sariyar, 2012), **MiscPsycho** (Doran, 2010), **cba** (Buchta and Hahsler, 2013), and **Mkmisc** (Kohl, 2013) offer interfaces to the Levenshtein distance, while **deducorrect** (van der Loo et al., 2011) implements the restricted Damerau-Levenshtein distance¹. Although the distances implemented in these packages are all based on (an extension of) the Levenshtein distance, interfaces may vary: **RecordLinkage** takes character data as input and computes byte-wise distances ignoring the possibility of multibyte character encoding. Packages **MkMisc** and **deducorrect** do exactly the opposite and always compare strings character-wise. The **cba** package can compute bitwise distances between character data as well as between lists of integers, leaving a translation from (possibly multibyte character) data to integers to the user. Finally, the **textcat** package of Hornik et al. (2013), which implements q -gram based text classification methods of Cavnar and Trenkle (1994), offers a range of q -gram based distances which can be computed on a character-wise or byte-wise basis.

There are several packages that offer string-like distance functions applied to specialized objects. The **TraMineR** package (Studer et al., 2011) for example implements various distance measures on

¹The **vwr** package (Keuleers, 2013) used to offer its own implementation of string distances but currently depends on **stringdist**.

“state sequence objects” including the [Hamming \(1950\)](#) distance, the Longest Common Substring distance and more. Although it may be possible for users to translate character strings representing text to a state sequence object, this obviously means using **TraMineR** for purposes for which it was not intended, which is undesirable. Similarly, the Hamming distance is implemented in at least five more packages for object types ranging from presentations of graphs ([Butts, 2013](#)), to rank vectors ([Grimonprez, 2013](#)) to phylogenetic data ([Schliep and Paradis, 2013](#)).

The fact that several authors have taken the trouble to implement similar functionality shows that the ability to compute offline string distance measures is a fairly basic need across fields that deserves specialized attention. Moreover, the variety of input types and interpretations thereof make reuse and comparison of these implementations less than transparent.

The **stringdist** package presented in this paper aims to help users by offering a uniform interface to a number of well-known string distance measures where special values and (non)interpretation of character encoding are handled transparently and consistently across metrics. The package re-implements some previously available distances and adds a number of distance measures which were according to this author’s best knowledge previously unavailable in R. Distances implemented by the package include edit-based distances (Hamming, generalized Levenshtein, Longest Common Substring, optimal string alignment, and generalized Damerau-Levenshtein), q -gram based distances (q -gram, Jaccard, and cosine) and the heuristic Jaro and Jaro-Winkler distances. Of these distances, at least the generalized Damerau-Levenshtein distance and the Jaccard distance appear to be new in the context of character strings. Core distance functions have been implemented as a C library for speed and exposed to R. Based on this functionality, the package implements approximate matching equivalents of base R’s table lookup functions `match` and `%in%`. A convenience function that lists q -gram occurrences is included as well.

The rest of this paper is organized as follows. In the next section we give a quick overview of the package’s main functionality and discuss how special values and character encodings are handled. The section after that is devoted to a concise description of the distance functions offered by **stringdist**. We will work out simple examples, and point out some properties of the various metrics. The section is aimed to serve as a reference to users of the package. We end with some conclusions and an outlook.

The stringdist package

The package offers two basic kinds of functionality: computing string comparison metrics and table lookup by approximate string matching. String distances can be computed with the function `stringdist` or `stringdistmatrix`. Both take at least two character vectors as arguments, but the former computes element-wise string distances where the shorter argument is recycled, and the latter returns the distance matrix.

```
> stringdist('foo', c('fu','bar',NA))
[1] 2 3 NA

> stringdistmatrix(c('foo','bar'), c('fu','bar',NA))
      [,1] [,2] [,3]
[1,]    2    3  NA
[2,]    3    0  NA
```

Here, the default distance is the optimal string alignment distance which will be discussed in more detail below. To specify another algorithm one simply sets the `method` option. For example, to compute the Levenshtein distance one uses the following.

```
> stringdist('foo', 'bar', method='lv')
```

String distance functions have two possible special output values. NA is returned whenever at least one of the input strings to compare is NA and Inf is returned when the distance between two strings is undefined according to the selected algorithm. For example, the Hamming distance is undefined when comparing strings of different length.

```
> stringdist('fu', 'foo', method='hamming')
[1] Inf
```

The functions `stringdist` and `stringdistmatrix` have very similar interfaces, allowing users to select from nine different string distance algorithms, define weights (depending on the chosen distance) and more. The `stringdistmatrix` function has extra options that allow users to parallelize calculation of the distance matrix over columns. Users may specify the number of cores to use or pass a `cluster` object as generated by `makeCluster` of the **parallel** package. For example, the command

```
> stringdistmatrix(c('foo','bar'), c('fu','bar',NA), ncores=3)
```

distributes calculation of the columns of the distance matrix over three local cores.

Approximate matching can be done with the functions `amatch` and `ain`. Function `amatch(x, table)` finds the closest match of elements of `x` in `table`. When multiple equivalent matches are found, the first match is returned. A call to `ain(x, table)` returns a logical vector indicating which elements of `x` were (approximately) matched in `table`. Both `amatch` and `ain` have been designed to approach the behaviour of R's native `match` and `%in%` functionality as much as possible. By default `amatch` and `ain` locate exact matches, just like `match`. This may be changed by increasing the maximum string distance between the search pattern and elements of the lookup table.

```
> amatch('fu', c('foo','bar'), maxDist=2)
[1] 1
> ain('fu', c('foo','bar'), maxDist=2)
[1] TRUE
```

The default distance function is again the optimal string alignment distance, but this can be controlled by altering the `method` option. Here, the string `'fu'` matches approximately with `'foo'` since in the default metric the difference is two operations (replace one `'u'` and add an `'o'`). By default, `NA` is matched with `NA`, as in R's native `match` function.

```
> amatch(NA, c('foo',NA))
[1] 2
```

Unlike in `match`, this may be switched off by setting `matchNA=FALSE`.

```
> amatch(NA, c('foo',NA), matchNA=FALSE)
[1] NA
```

Like in `match`, the `nomatch` option controls the output when no match is found.

```
> amatch(NA, c('foo',NA), matchNA=FALSE, nomatch=0)
[1] 0
```

Character encoding

A character encoding system defines how each character in an alphabet is represented as a byte or sequence of bytes in computer memory. Over the past decades many encoding systems have been developed, and currently several encoding standards are widely in use. It is a rather unfortunate state of affairs that encoding standards have evolved to a point where it is impossible to determine the used encoding from a text file with certainty from its contents, although command-line utilities like `'file'` (under Unix-alikes) often do a good job at guessing it.

By default, the character encoding used internally by R depends on the native encoding of the system on which it runs. This means that when one needs to read a text file that is stored in a different encoding (for example because it was produced on a different operating system), the file's encoding has to be explicitly specified. For example, in `read.table` the input encoding can be specified using the `fileEncoding` argument. Upon reading the file, R will attempt to translate input from the specified encoding to the internal encoding native to the operating system it is running on.

As a consequence, reading the same file into R will not always yield the same sequence of bytes internally on each system. Only when one is certain that the input consists solely of characters from the ASCII character set will the internal representation be guaranteed to be the same across systems. Most of R's native functions hide the character representation effectively from the user. For example, `nchar` counts the number of actual characters, not the number of bytes used to represent it (although it has an option to count the number of bytes as well).

Like R's native functions, all functions of the **stringdist** package are designed to yield the same result regardless of the internal encoding used. Like in the `adist` function, this is done by converting strings first to `utf8`, and then to an integer representation. The integer representation of strings are passed to the underlying C-routines. At the moment, this double conversion is the only guaranteed way to handle character strings independent of internal encoding. The main reason is that R depends on external libraries for character re-encoding, and those libraries are different across operating systems on which R is supported.

The re-encoding causes an overhead of up to a factor of three or four, when computing distances between character strings consisting of 5-25 characters. The overhead is almost completely due to the conversion to integer and its relative importance decreases with string length. If one is certain that the input strings are restricted to the ASCII character set or when accuracy or cross-platform

independence are of less importance, one can pass `useBytes=TRUE` to avoid re-encoding. In that case the distance between the underlying byte sequences are returned. This option mimics the option that is available in some of R's native functions, including `adist`. Below is an example demonstrating the difference.

```
> stringdist('Motorhead', 'Motörhead')
[1] 1
> stringdist('Motorhead', enc2utf8('Motörhead'), useBytes=TRUE)
[1] 2
```

Here, the default string distance algorithm is the optimal string alignment distance, which counts the number of insertions, deletions, substitutions and transpositions necessary to turn 'Motörhead' into 'Motorhead'. In the first case the letter 'ö' is recognized as a single character, so the distance corresponds to a single substitution. In the second case a byte-wise comparison is used while making sure that 'Motörhead' is stored in utf8 encoding. In this case, the utf8 'ö' is stored as a single byte and 'ö' as two bytes, and here the distance is determined by deleting one byte and substituting another.

It should be mentioned that there are a number of characters that have multiple utf8 representations (this is called 'unicode equivalence'). For example, the 'ö' can be represented by a two-byte unicode character as in the above example or by a single-byte 'o', followed by a two-byte (otherwise invisible) modifying character that specifies the umlaut. If one compares the two versions of 'ö' with either `stringdist` or R's native `adist`, the result will be nonzero regardless whether one compares byte-wise or not. The solution is to normalize unicode-encoded strings to either representation before any comparison is made. At the moment, no tools seem to be available from within R but open source commandline tools like `uconv` (Utterström and Arrouye) can be used to normalize utf8-encoded text prior to reading into R.

String distance functions

A *string* is a finite concatenation of symbols (characters) from a finite alphabet. We will follow the usual notation and denote a finite alphabet with Σ where the number of elements in Σ is denoted $|\Sigma|$. The q -fold Cartesian product $\Sigma \times \dots \times \Sigma$ is denoted Σ^q , and the set of all finite strings that can be composed from characters in Σ is denoted Σ^* . The empty string, denoted ϵ , is also a member of this set. We will denote a general string with s , t , or u and the length of the string, measured as the number of characters, with $|s|$. For example: take for Σ the 26-member lower-case Latin alphabet, and $s = \text{'foo'}$. We have $|s| = 3$, $s \in \Sigma^3$ and $s \in \Sigma^*$. Individual characters of a string are indicated with a subscript so in the previous example we have $s_1 = \text{'f'}$ and $s_2 = s_3 = \text{'o'}$. A subsequence is indicated with subscript $m : n$, so in the example $s_{1:2} = \text{'fo'}$. We also use the convention that $s_{m:n} = \epsilon$ whenever $n < m$.

Formally, for a function d to be a *distance function* on Σ^* it has to obey the following properties.

$$\text{nonnegativity} \quad d(s, t) \geq 0 \quad (1a)$$

$$\text{identity} \quad d(s, t) = 0 \text{ only when } s = t \quad (1b)$$

$$\text{symmetry} \quad d(s, t) = d(t, s) \quad (1c)$$

$$\text{triangle inequality} \quad d(s, u) \leq d(s, t) + d(t, u), \quad (1d)$$

with s , t , and u strings. However, as will be pointed out below, many string metrics do not have all these properties. For example, distances based on q -grams do not necessarily obey the identity property and weighted versions of the Levenshtein distance are not necessarily symmetric.

Distances based on edit operations

Distances based on edit operations count the number of basic operations necessary to turn one string into another. Edit-like distances can be categorized based on what operations are allowed. The distances discussed below allow one or more of the following operations.

- Substitution of a character, as in $\text{'foo'} \rightarrow \text{'boo'}$.
- Deletion of a character, as in $\text{'foo'} \rightarrow \text{'oo'}$.
- Insertion of a character, as in $\text{'foo'} \rightarrow \text{'floo'}$.
- Transposition of two adjacent characters, as in $\text{'foo'} \rightarrow \text{'of o'}$.

For distances allowing more than a single operation it may be meaningful to assign weights to the different operations, for example to make a transposition contribute less to the distance than character

substitution. Edit-based distances for which such weights can be defined are usually referred to as *generalized distances* (Boytssov, 2011).

The simplest edit distance is the *Hamming distance* (Hamming, 1950), which allows only character substitutions and is therefore only defined for strings of equal length. It is however common to define the Hamming distance to be infinite for strings of different length (Navarro, 2001), so formally we have

$$d_{\text{hamming}}(s, t) = \sum_{i=1}^{|s|} [1 - \delta(s_i, t_i)] \text{ if } |s| = |t| \text{ and } \infty \text{ otherwise.} \quad (2)$$

Here, $\delta(s_i, t_j) = 1$ if $s_i = t_j$ and 0 otherwise. The Hamming distance obeys all the properties stated in Eq. (1). When s and t are of equal length, the maximum value is $|s|$. With the **stringdist** package, the Hamming distance can be computed as follows.

```
> stringdist(c('foo', 'fu'), 'bar', method='hamming')
[1] 3 Inf
```

The *Longest Common Substring* distance d_{lcs} (Needleman and Wunsch, 1970) counts the number of deletions and insertions necessary to transform one string into another. It can be recursively defined as

$$d_{\text{lcs}}(s, t) = \begin{cases} 0 & \text{if } s = t = \varepsilon, \\ d_{\text{lcs}}(s_{1:|s|-1}, t_{1:|t|-1}) & \text{if } s_{|s|} = t_{|t|}, \\ 1 + \min\{d_{\text{lcs}}(s_{1:|s|-1}, t), d_{\text{lcs}}(s, t_{1:|t|-1})\} & \text{otherwise.} \end{cases} \quad (3)$$

The longest common substring distance obeys all properties of Eq. (1). It varies between 0 and $|s| + |t|$ where the maximum is achieved when s and t have no characters in common. With the **stringdist** package it can be computed by setting `method='lcs'`.

```
> stringdist('leia', 'leela', method='lcs')
[1] 3
```

Here, the distance equals 3 since it takes two deletions and one insertion to turn 'leela' into 'leia':

$$\text{leela} \xrightarrow{\text{del. e}} \text{lela} \xrightarrow{\text{del. l}} \text{lea} \xrightarrow{\text{ins. i}} \text{leia}.$$

The above example also shows that in general there is no unique shortest path between two strings: one could for example reverse the order of the first two deletions to obtain a second path with total weight 3.

As suggested by its name, the lcs-distance has a second interpretation. With the *longest common substring* we mean the longest sequence formed by pairing characters from s and t while keeping their order intact. The lcs-distance is then the number of unpaired characters over both strings. In the above example this can be visualized as follows.

$$\begin{array}{ccc} \begin{array}{c} \text{l e e l a} \\ | \quad | \quad / \\ \text{l e i a} \end{array} & \text{or, equivalently} & \begin{array}{c} \text{l e e l a} \\ | \quad / \quad / \\ \text{l e i a} \end{array} \end{array}$$

In both cases, the characters 'e', 'l' and 'i' remain unpaired, independent of whether we start pairing from the beginning (left case) or the end of the string (right case), yielding a distance value of three.

The *generalized Levenshtein distance* d_{lv} is computed by counting the weighted number of insertions, deletions and substitutions necessary to turn one string into another. Like the lcs distance it permits a recursive definition.

$$d_{\text{lv}}(s, t) = \begin{cases} 0 & \text{if } s = t = \varepsilon \\ \min\{ \\ \quad d_{\text{lv}}(s, t_{1:|t|-1}) + w_1, \\ \quad d_{\text{lv}}(s_{1:|s|-1}, t) + w_2, \\ \quad d_{\text{lv}}(s_{1:|s|-1}, t_{1:|t|-1}) + [1 - \delta(s_{|s|}, t_{|t|})]w_3 \\ \} & \text{otherwise.} \end{cases} \quad (4)$$

Here, w_1 , w_2 and w_3 are the nonnegative penalties for deletion, insertion, and substitution when turning t into s . The generalized Levenshtein distance is also implemented by R's native `adist` function, and with our package it can be computed as follows.

```
> stringdist('leela', 'leia', method='lv')
[1] 2
```

The extra flexibility with respect to the lcs (and hamming) distance yields a smaller distance value since we need but a single deletion and a single substitution:

$$\text{leela} \xrightarrow[+1]{\text{del. e}} \text{lela} \xrightarrow[+1]{\text{sub. l} \rightarrow \text{i}} \text{leia},$$

where we denote the penalty for each operation below the arrow.

The generalized Levenshtein distance obeys the properties of Eq. (1) except when $w_1 \neq w_2$, in which case the symmetry property is lost. However, it remains symmetric under simultaneous reversal of s and t and w_1 and w_2 since the number of deletions necessary going from t to s is equal to the number of insertions necessary going from s to t . This is illustrated by the following example.

```
> stringdist('leia', 'leela', method='lv', weight=c(1,0.1,1))
[1] 2
> stringdist('leia', 'leela', method='lv', weight=c(0.1,1,1))
[1] 1.1
> stringdist('leela', 'leia', method='lv', weight=c(1,0.1,1))
[1] 1.1
```

In the first line we get a distance of two since no insertions (of weight 0.1) are involved when going from 'leela' to 'leia', as shown in our previous example. The second and third example may schematically be represented as follows.

$$\begin{aligned} \text{leela} &\xrightarrow[+0.1]{\text{del. e}} \text{lela} \xrightarrow[+1]{\text{sub. l} \rightarrow \text{i}} \text{leia} \\ \text{leia} &\xrightarrow[+1]{\text{sub. i} \rightarrow \text{l}} \text{lela} \xrightarrow[+0.1]{\text{ins. e}} \text{leela}. \end{aligned}$$

In other words, reversing w_1 and w_2 is the same as reversing arguments s and t .

The *optimal string alignment* distance d_{osa} is a straightforward extension of the Levenshtein distance that allows for transpositions of adjacent characters:

$$d_{\text{osa}}(s, t) = \begin{cases} 0 & \text{if } s = t = \varepsilon \\ \min\{ & \\ \quad d_{\text{osa}}(s, t_{1:|t|-1}) + w_1, & \\ \quad d_{\text{osa}}(s_{1:|s|-1}, t) + w_2, & \\ \quad d_{\text{osa}}(s_{1:|s|-1}, t_{1:|t|-1}) + [1 - \delta(s_{|s|}, t_{|t|})]w_3, & \\ \quad d_{\text{osa}}(s_{1:|s|-2}, t_{1:|t|-2}) + w_4 & \text{if } s_{|s|} = t_{|t|-1}, s_{|s|-1} = t_{|t|} \\ \} & \text{otherwise,} \end{cases} \quad (5)$$

where w_4 is the penalty for a transposition and w_1, w_2 and w_3 were defined under Eq. (4). The optimal string alignment distance is the default distance function for `stringdist`, `stringdistmatrix`, `amatch` and `ain`. Unlike the Hamming, lcs, and Levenshtein distances, the optimal string alignment distance does not obey the triangle inequality. This is demonstrated by the following example, taken from Boytsov (2011).

```
> stringdist('ba', 'ab') + stringdist('ab', 'acb')
[1] 2
> stringdist('ba', 'acb')
[1] 3
```

The reason is that the optimal string alignment distance edits each substring maximally once while recursing through the strings. The above distances should be interpreted as

$$\begin{aligned} \text{ba} &\xrightarrow[+1]{\text{swap b,a}} \text{ab} + \text{ab} \xrightarrow[+1]{\text{ins. c}} \text{acb} \\ \text{ba} &\xrightarrow[+1]{\text{del. b}} \text{a} \xrightarrow[+1]{\text{ins. c}} \text{ac} \xrightarrow[+1]{\text{ins. b}} \text{acb}. \end{aligned}$$

In the last case, the shortcut that can be taken by swapping 'b' and 'a' and then inserting 'c' would force the same substring to be edited twice. Because of this restriction, the optimal string alignment distance is also referred to as the *restricted Damerau-Levenshtein distance* although on the web, it seems fairly often to be mistaken for the actual *Damerau-Levenshtein distance*. The latter distance metric does allow for multiple edits on the same substring and is a real metric in the sense of Eq. (1).

A recursive definition for the full Damerau-Levenshtein distance was first given by Lowrance and Wagner (1975). Their definition replaces the simple swap in the last line of Eq. (5) with a minimization over possible transpositions between the current character and all untreated characters, where the cost

of a transposition increases with the distance between transposed characters. In the **stringdist** package a weighted version of the full Damerau-Levenshtein distance is implemented which is defined as follows.

$$d_{dl}(s, t) = \begin{cases} 0 & \text{if } s = t = \varepsilon \\ \min\{ & \\ d_{dl}(s, t_{1:|t|-1}) + w_1, & \\ d_{dl}(s_{1:|s|-1}, t) + w_2, & \\ d_{dl}(s_{1:|s|-1}, t_{1:|t|-1}) + [1 - \delta(s_{|s|}, t_{|t|})]w_3, & \\ \min_{(i,j) \in \Lambda} d_{dl}(s_{1:i-1}, t_{1:j-1}) + [(|s| - i) + (|t| - j) - 1]w_4 & \\ \} & \text{otherwise,} \end{cases} \quad (6)$$

where the minimization in the last line is over

$$\Lambda = \{(i, j) \in \{1, \dots, |s|\} \times \{1, \dots, |t|\} : s_{|s|} = t_j, s_i = t_{|t|}\}.$$

If $w_1 = w_2 = w_3 = w_4 = 1$ the original Damerau-Levenshtein distance is obtained. Although it is a trivial generalization of the original distance described by [Lowrance and Wagner \(1975\)](#), this implementation of a generalized Damerau-Levenshtein distance appears to be new in literature.

For the **stringdist** package, the C code for this particular distance is based on a routine developed by [Logan \(2013\)](#). The original code has been adapted to reduce the number of memory allocation calls and to include the weights. The Damerau-Levenshtein distance can be computed with the `method='dl'` directive.

```
> stringdist('ba', 'acb', method='dl')
[1] 2
```

Here, the distance equals two, indicating that the path

$$ba \xrightarrow{\text{swap } b,a} ab \xrightarrow{\text{ins. } c} acb,$$

is indeed included in the minimization defining the distance.

The maximum distance between two strings s and t , as measured by either the Levenshtein, optimal string alignment, or Damerau-Levenshtein distance is $\max\{|s|, |t|\}$. However, as the number of possible edit operations increases, the possible number of paths between two strings increases, allowing for possibly smaller distances between strings. Therefore, relations between the distance measures described above can be summarized as follows.

$$\left. \begin{aligned} \infty &\geq |s| \geq d_{\text{hamming}}(s, t) \\ |s| + |t| &\geq d_{\text{lcs}}(s, t) \\ \max\{|s|, |t|\} &\end{aligned} \right\} \geq d_{\text{lv}}(s, t) \geq d_{\text{osa}}(s, t) \geq d_{\text{dl}}(s, t) \geq 0. \quad (7)$$

Since the Hamming and lcs distance have no basic edits in common, there is no order relation between their values. The upper limit $|s|$ on the Hamming distance only holds when $|s| = |t|$.

All edit-based distances except the Hamming distance have been implemented using the well-known dynamic programming technique that runs in $\mathcal{O}(|s||t|)$ time. For the Hamming distance, both the time and memory consumption are $\mathcal{O}(|s|)$. For the other edit-based distances the memory consumption is $\mathcal{O}(|s||t|)$, where for the Damerau-Levenshtein some extra memory is used (growing with the number of unique characters in s and t). Finally, we refer the reader to the papers of [Boytssov \(2011\)](#) and [Navarro \(2001\)](#) for a thorough review of edit-based distances in text search or dictionary lookup settings respectively.

Distances based on q -grams

A q -gram is a string consisting of q consecutive characters. The q -grams associated with a string s are obtained by sliding a window of q characters wide over s and registering the occurring q -grams. For example, the digrams associated with 'foo' are 'fo' and 'oo'. Obviously, this procedure fails when $q > |s|$ or $q = 0$. For this reason we define the following edge cases for all distances $d(s, t; q)$ of the **stringdist** package that are based on comparing q -gram occurrence:

$$d(s, t; q) = \infty, \text{ when } q > \min\{|s|, |t|\} \quad (8a)$$

$$d(s, t; 0) = \infty, \text{ when } |s| + |t| > 0 \quad (8b)$$

$$d(\varepsilon, \varepsilon; 0) = 0. \quad (8c)$$

A simple distance metric between two strings is obtained by listing unique q -grams in two strings and compare which ones they have in common. Indeed, if we write $Q(s; q)$ to indicate the unique set of q -grams occurring in s , the *Jaccard distance* is written as

$$d_{\text{jaccard}}(s, t; q) = 1 - \frac{|Q(s; q) \cap Q(t; q)|}{|Q(s; q) \cup Q(t; q)|} \quad (9)$$

where the vertical bars ($|\cdot|$) indicate set cardinality. The Jaccard distance varies from 0 to 1 where 0 corresponds to full overlap, *i.e.* $Q(s; q) = Q(t; q)$, and 1 to no overlap, *i.e.* $Q(s; q) \cap Q(t; q) = \emptyset$. As an example, consider result of the following Jaccard distance calculation with the **stringdist** package.

```
> stringdist('leia', 'leela', method='jaccard', q=2)
[1] 0.8333333
```

It is easily checked that $Q('leia'; 2) = \{'le', 'ei', 'ia'\}$ and $Q('leela'; 2) = \{'le', 'ee', 'el', 'la'\}$, so the distance is computed as $1 - \frac{1}{6} \approx 0.83$.

The q -gram distance is obtained by tabulating the q -grams occurring in the two strings and counting the number of q -grams that are not shared between the two. This may formally be denoted as follows.

$$d_{\text{qgram}}(s, t; q) = \|v(s; q) - v(t; q)\|_1 = \sum_{i=1}^{|\Sigma|^q} |v_i(s; q) - v_i(t; q)|. \quad (10)$$

Here, $v(s; q)$ is a nonnegative integer vector of dimension $|\Sigma|^q$ whose coefficients represent the number of occurrences of every possible q -gram in s . Eq. (10) defines the q -gram distance between two strings s and t as the L_1 distance between $v(s; q)$ and $v(t; q)$. Observe that one only needs to store and count the actually occurring q -grams to evaluate the above sum.

With the **stringdist** package, q -gram distances are computed as follows.

```
> stringdist('leia', 'leela', method='qgram', q=1)
[1] 3
> stringdist('leia', 'leela', method='qgram', q=2)
[1] 5
> stringdist('leia', 'leela', method='qgram', q=5)
[1] Inf
```

The 1-gram distance between 'leia' and 'leela' equals 3: counting the 1-grams (individual characters) of the two strings shows that the 'i' of 'leia' and the second 'e' and 'l' of 'leela' are unmatched. The reader may verify that the 2-gram distance between 'leia' and 'leela' equals 5. In the third example, **stringdist** returns Inf since one of the compared strings has less than 5 characters.

The maximum number of different q -grams in a string s is $|s| - q + 1$, therefore $|s| + |t| - 2q + 2$ is an upper bound on the q -gram distance, occurring when s and t have no q -grams in common. See [Boytsov \(2011\)](#) and references therein for bounds on d_{qgram} in terms of edit-based distances.

Now that we have defined the q -gram distance in terms of vectors, any distance function on (integer) vector spaces can in principle be applied. The **stringdist** package also includes the *cosine distance*, which is defined as

$$d_{\text{cos}}(s, t; q) = 1 - \frac{v(s; q) \cdot v(t; q)}{\|v(s; q)\|_2 \|v(t; q)\|_2}, \quad (11)$$

where $\|\cdot\|_2$ indicates the standard Euclidean norm. The cosine distance equals 0 when $s = t$ and 1 when s and t have no q -grams in common. It should be interpreted as a measure of the angle between $v(s; q)$ and $v(t; q)$ since the second term in Eq. (11) represents the cosine of the angle between the two vectors. It can be computed as follows.

```
> stringdist('leia', 'leela', method='cosine', q=1)
[1] 0.1666667
```

Indeed, defining $\Sigma = \{'a', 'e', 'i', 'l'\}$, we have $v('leia'; 1) = (1, 1, 1, 1)$ and $v('leela'; 1) = (1, 2, 0, 2)$ giving a distance of $1 - \frac{5}{2 \cdot 3} \approx 0.17$.

The three q -gram based distances mentioned above are nonnegative and symmetric. The Jaccard and q -gram distance can be written as a distance on a vector space and obey the triangle inequality as well. The cosine distance does not satisfy the triangle inequality. None of the q -gram based distances satisfy the identity condition because both $Q(s; q)$ and $v(s; q)$ are many-to-one functions. As an example observe that $Q('ab'; 1) = Q('ba'; 1)$ and $v('ab'; 1) = v('ba'; 1)$ so $d_{\text{jaccard}}('ab', 'ba'; 1) = d_{\text{qgram}}('ab', 'ba'; 1) = d_{\text{cos}}('ab', 'ba'; 1) = 0$. In other words, a q -gram based distance of zero does not guarantee that $s = t$. For a more general account of invariance properties of the $v(s; q)$ the reader is referred to [Ukkonen \(1992\)](#).

To allow the user to define their own q -gram based metrics, the package includes the function `qgrams`. This function takes an arbitrary number of (named) character vectors as input, and returns an array of labeled q -gram counts. Here's an example with three character vectors.

```
> qgrams(
+   x = c('foo', 'bar', 'bar'),
+   y = c('fu', 'bar'),
+   z = c('foobar'),
+   q = 2 )
      fo oo fu ob ba ar
x  1  1  0  0  2  2
y  0  0  1  0  1  1
z  1  1  0  1  1  1
```

At the moment, q -gram counting is implemented by storing only the encountered q -grams, rather than representing $v(s; q)$ completely. This avoids the need for $\mathcal{O}(|\Sigma|^q)$ storage. Encountered q -grams are stored in a binary tree structure yielding $\mathcal{O}(|Q(s; q)|)$ memory and $\mathcal{O}[(|s| - q - 1) \log |Q(s; q)|]$ time consumption.

Heuristic distance measures

The *Jaro distance* was originally developed at the U.S. Bureau of the Census for the purpose of linking records based on inaccurate text fields. Its first public description appeared in a user manual (Jaro, 1978) which might explain why it is not very wide-spread in computer science literature. It has however been successfully applied to statistical matching problems concerning fairly short strings; typically name and address data [see e.g. Jaro (1989)].

The reasoning behind the Jaro distance is that character mismatches and transpositions are caused by typing-errors but matches between remote characters are unlikely to be caused by a typing error. It therefore measures the number of matching characters between two strings that are not too many positions apart and adds a penalty for matching characters that are transposed. It is given by

$$d_{\text{jaro}}(s, t) = \begin{cases} 0 & \text{when } s = t = \varepsilon \\ 1 & \text{when } m = 0 \text{ and } |s| + |t| > 0 \\ 1 - \frac{1}{3} \left(w_1 \frac{m}{|s|} + w_2 \frac{m}{|t|} + w_3 \frac{m-T}{m} \right) & \text{otherwise.} \end{cases} \quad (12)$$

Here, the w_i are adjustable weights but in most publications they are chosen equal to 1. Furthermore, m is the number of characters that can be matched between s and t . Supposing that $s_i = t_j$ they are considered a match only when

$$|i - j| < \left\lfloor \frac{\max\{|s|, |t|\}}{2} \right\rfloor,$$

and every character in s can be matched only once with a character in t . Finally, if s' and t' are substrings of s and t obtained by removing the nonmatching characters, then T is the number of transpositions necessary to turn s' into t' . Here, nonadjacent transpositions are allowed.

With `stringdist`, the Jaro-distance can be computed as follows.

```
> stringdist('leia', 'leela', method='jw')
[1] 0.2166667
```

Here, the number of matching characters equals three, and no transpositions are necessary yielding a distance of $1 - \frac{1}{3}(\frac{3}{4} + \frac{3}{5} + 1) = \frac{13}{60} \approx 0.22$. When $w_1 = w_2 = w_3 = 1$, the Jaro distance ranges between 0 and 1, where 0 corresponds to $s = t$ and 1 indicates a complete dissimilarity with $m = T = 0$.

Winkler (1990) extended the Jaro distance by incorporating an extra penalty for character mismatches in the first four characters. The *Jaro-Winkler distance* is given by

$$d_{\text{jw}}(s, t) = d_{\text{jaro}}(s, t)[1 - p\ell(s, t)], \quad (13)$$

where $\ell(s, t)$ is the length of the longest common prefix, up to a maximum of four characters and p is a user-defined weight. We demand that $p \in [0, \frac{1}{4}]$ to make sure that $0 \leq d_{\text{jw}}(s, t) \leq 1$. The factor p determines how strongly differences between the first four characters of both strings determine the total distance. If $p = 0$, the Jaro-Winkler distance reduces to the Jaro distance and all characters contribute equally to the distance function. If $p = \frac{1}{4}$, the Jaro-Winkler distance is equal to zero, even if only the first four characters differ. The reasoning is that apparently, people are less apt to make mistakes in the first four characters or perhaps they are more easily noted, so differences in the first four characters point to a larger probability of two strings being actually different. Winkler (1990) and Cohen et al. (2003) use a value of $p = 0.1$ and report better results in a statistical matching benchmark

than with $p = 0$. The default value of p for the `stringdist` function with `method='jw'` is 0 so by altering it, the Jaro-Winkler distance is obtained.

```
> stringdist('leia', 'leela', method='jw', p=0.1)
[1] 0.1733333
```

Here, we have $\ell('leia', 'leela') = 2$ so the Jaro-Winkler distance is computed as $\frac{13}{60}(1 - \frac{2}{10}) \approx 0.17$.

It is easy to see from Eqs. (12) and (13) that conditional on $w_1 = w_2 = w_3 = 1$, the Jaro and Jaro-Winkler distance are nonnegative, symmetric and obey the identity property. However, the triangle inequality is not satisfied by these distances. As an example consider $s = 'ab'$, $t = 'cb'$ and $u = 'cd'$. Since s and u have no characters in common we have $d_{\text{jaro}}(s, u) = 1$, while $d_{\text{jaro}}(s, t) = d_{\text{jaro}}(t, u) = \frac{1}{3}$ so in this case $d_{\text{jaro}}(s, u)$ is larger than $d_{\text{jaro}}(s, t) + d_{\text{jaro}}(t, u)$. It is not difficult to verify that the Jaro-Winkler distance fails the triangle inequality for the same example, for any $p \in [0, \frac{1}{4}]$.

The C-implementation of the Jaro and Jaro-Winkler distance take $\mathcal{O}(|s||t|)$ time and $\mathcal{O}(\max\{|s|, |t|\})$ memory.

What metric to use?

In the end the choice depends on the application, but there are some general considerations. The choice between an edit-based or heuristic metric on one hand or a q -gram based distance on the other, is to an extent prescribed by string length. Contrary to edit-based or heuristic metrics, q -gram based metrics can easily be computed between very long text strings since the number of q -grams encountered in natural language (for say, $q \geq 3$) is usually much less than the q -grams allowed by the alphabet. The choice of edit-based distance depends mostly on the needed accuracy. For example, in a dictionary lookup where differences between matched and dictionary items are small, an edit distance that allows for more types of edit operations (like the optimal string alignment or Damerau-Levenshtein distance) may give better results. The heuristic Jaro- and Jaro-Winkler distances were designed with human-typed, relatively short strings in mind, so their area of application is clear.

Summary and conclusions

The `stringdist` package offers, for the first time in R, a number of popular string distance functions through a consistent interface while transparently handling or ignoring the underlying character encoding scheme. The package offers interfaces to C-based string distance algorithms that either recycle elements or return the full distance matrix. The same algorithms are used in approximate string matching versions of R's native `match` and `%in%` functions: `amatch` and `ain` respectively.

In this paper we have given a concise description of the available distance functions in terms of their mathematical definitions and showed how to compute them. The algorithmic complexity of the current implementation in terms of computational time and memory consumption was mentioned as well.

In the future, we expect to make the C-library available for export to other languages and to reduce the memory consumption for some of the algorithms.

Acknowledgements

The author is grateful to Dr. Rob van Harreveld for carefully reading the manuscript.

Bibliography

- A. Borg and M. Sariyar. *RecordLinkage: Record Linkage in R*, 2012. URL <http://CRAN.R-project.org/package=RecordLinkage>. R package version 0.4-1. [p111]
- L. Boytsov. Indexing methods for approximate dictionary searching: comparative analyses. *ACM Journal of Experimental Algorithmics*, 16:1–86, 2011. [p115, 116, 117, 118]
- C. Buchta and M. Hahsler. *cba: Clustering for Business Analytics*, 2013. URL <http://CRAN.R-project.org/package=cba>. R package version 0.2.12. [p111]
- C. T. Butts. *sna: Tools for Social Network Analysis*, 2013. URL <http://CRAN.R-project.org/package=sna>. R package version 2.3-1. [p112]

- W. Cavnar and J. Trenkle. N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, 1994. [p111]
- W. Cohen, P. Ravikumar, and F. Fienberg. A comparison of string distance metrics for name-matching tasks. In *Proceedings of the IJCAI-03 workshop on Information Integration on the Web*, pages 73–78, 2003. URL <http://www.isi.edu/integration/workshops/ijcai03/proceedings.htm>. [p119]
- H. C. Doran. *MiscPsycho: Miscalleaneous Psychometric Analyses*, 2010. URL <http://CRAN.R-project.org/package=MiscPsycho>. R package version 1.6. [p111]
- Q. Grimonprez. *Rankcluster: Model-based clustering for multivariate partial ranking data*, 2013. URL <http://CRAN.R-project.org/package=Rankcluster>. R package version 0.90.3. [p112]
- R. Hamming. Error detecting and error correcting codes. *The Bell system technical journal*, 29:147–160, 1950. [p112, 115]
- K. Hornik, P. Mair, J. Rauch, W. Geiger, C. Buchta, and I. Feinerer. The textcat package for n-gram based text categorization in R. *Journal of Statistical Software*, 52(6):1–17, 2 2013. ISSN 1548-7660. URL <http://www.jstatsoft.org/v52/i06>. [p111]
- M. Jaro. *UNIMATCH: A record linkage system: User manual*. United States bureau of the census, 1978. pp. 103-108. [p119]
- M. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida. *Journal of the American Statistical Association*, 84:414–420, 1989. [p119]
- A. Karatzoglou, A. Smola, K. Hornik, and A. Zeileis. kernlab – an S4 package for kernel methods in R. *Journal of Statistical Software*, 11(9):1–20, 2004. URL <http://www.jstatsoft.org/v11/i09/>. [p111]
- E. Keuleers. *vwr: Useful functions for visual word recognition research*, 2013. URL <http://CRAN.R-project.org/package=vwr>. R package version 0.3.0. [p111]
- M. Kohl. *MKmisc: Miscellaneous functions from M. Kohl*, 2013. URL <http://CRAN.R-project.org/package=MKmisc>. R package version 0.94. [p111]
- V. Laurikari. Efficient submatch addressing for regular expressions. Master’s thesis, Helsinki University of Technology, 2001. URL <https://github.com/laurikari/tre>. [p111]
- V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10:707–710, 1966. [p111]
- H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444, 2002. [p111]
- N. Logan. C code for Damerau-Levenshtein distance, 2013. URL <https://github.com/ugexe/Text--Levenshtein--Damerau--XS/blob/master/damerau-int.c>. Last accessed 2014-03-04. [p117]
- R. Lowrance and R. Wagner. An extension of the string-to-string correction problem. *Journal of the Association of Computing Machinery*, 22:177–183, 1975. [p116, 117]
- G. Navarro. A guided tour to approximate string matching. *ACM Computing surveys*, 33:31–88, 2001. [p115, 117]
- S. Needleman and C. D. Wunsch. A general method applicable to the search of similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970. [p115]
- K. Schliep and E. Paradis. *Phangorn: phylogenetic analyses in R*, 2013. URL <http://CRAN.R-project.org/package=Phangorn>. R package version 1.99-1. [p112]
- M. Studer, G. Ritschard, A. Gabadinho, and N. Müller. Discrepancy analysis of state sequences. *Sociological Methods and Research*, 40:471–510, 2011. [p111]
- E. Ukkonen. Approximate string-matching with q -grams and maximal matches. *Theoretical Computer Science*, 92:191–211, 1992. [p118]
- J. Utterström and Y. Arrouye. *uconv - convert data from one encoding to another (Linux man page)*. [p114]
- M. van der Loo, E. de Jonge, and S. Scholtus. *deducorrect: Deductive correction, deductive imputation, and deterministic correction.*, 2011. URL <https://github.com/markvanderloo/deducorrect>. R package version 1.3-5. [p111]

W. Winkler. String comparator metrics and enhanced decision rules in the Fellegi-Sunter model of record linkage. *Proceedings of the Section on Survey Research Methods (American Statistical Association)*, page 354–359, 1990. [p119]

Mark P.J. van der Loo

<http://www.markvanderloo.eu>

mark.vanderloo@gmail.com

RStorm: Developing and Testing Streaming Algorithms in R

by Maurits Kaptein

Abstract Streaming data, consisting of indefinitely evolving sequences, are becoming ubiquitous in many branches of science and in various applications. Computer scientists have developed streaming applications such as Storm and the S4 distributed stream computing platform¹ to deal with data streams. However, in current production packages testing and evaluating streaming algorithms is cumbersome. This paper presents **RStorm** for the development and evaluation of streaming algorithms analogous to these production packages, but implemented fully in R. **RStorm** allows developers of streaming algorithms to quickly test, iterate, and evaluate various implementations of streaming algorithms. The paper provides both a canonical computer science example, the streaming word count, and examples of several statistical applications of **RStorm**.

Introduction

Streaming data, consisting of indefinitely and possibly time-evolving sequences, are becoming ubiquitous in many branches of science (Chu et al., 2007; Michalak et al., 2012). The omnipresence of streaming data poses new challenges for statistics and machine learning. To enable user friendly development and evaluation of algorithms dealing with data streams this paper introduces **RStorm**.

Streaming learning algorithms can informally be described as algorithms which never “look back” to earlier data arriving at $t < t'$. Streaming algorithms provide a computationally efficient way to deal with continuous data streams by summarizing all historic data into a limited set of parameters. With the current growth of available data the development of reliable streaming algorithms whose behavior is well understood is highly important (Michalak et al., 2012). For a more formal description of streaming (or online) learning see Bottou (1998). Streaming analysis however provides both numerical as well as estimation challenges. Already for simple estimators, such as sample means and variances, multiple streaming algorithms can be deployed. For more complex statistical models, closed forms to exactly minimize popular cost functions in a stream are often unavailable.

Computer scientists recently developed a series of software packages for the streaming processing of data in production environments. Frameworks such as S4 by Yahoo! (Gopalakrishna et al., 2013), and Twitter’s Storm (Storm User Group, 2013) provide an infrastructure for *real-time* streaming computation of event-driven data (e.g., Babcock et al., 2002; Anagnostopoulos et al., 2012) which is scalable and reliable.

Recently, efforts have been made to facilitate easy testing and development of streaming processes within R for example with the **stream**. **stream** allows users of R to setup (or simulate) a data stream and specify data stream tasks to analyze the stream (Hahsler et al., 2014). While **stream** allows for the development and testing of streaming analysis in R, it does not have a strong link to current production environments in which streams can be utilized. Implementations of data streams in R analogous to production environments such as Twitter’s Storm are currently lacking. **RStorm** models the topology structure introduced by Storm², to enable development, testing, and graphical representation of streaming algorithms. **RStorm** is intended as a research and development package for those wishing to implement the analysis of data streams in frameworks outside of R, but who want to utilize R’s extensive plotting and data generating abilities to test their implementations. By providing an implementation of a data stream that is extremely comparable to the production code used in Storm, algorithms tested in R can easily be implemented in production environments.

Package RStorm: Counting words

In this section **RStorm** is introduced using the canonical streaming example used often for the introduction of Storm: a streaming word count. For **RStorm** the basic terminology and concepts from Storm³ are adapted, which are briefly explained before discussing the implementation of a streaming word count in **RStorm**. The aim of the streaming word count algorithm is to, given a stream of sentences – such as posts to a web service like Twitter – count the frequency of occurrence of each word. In Storm,

¹Not to be confused with the S4 object system used in R.

²This structure is very similar to the functioning of Yahoo!’s S4.

³The terms differ from those used by the S4 distributed stream computing platform, despite many similarities in functionality.

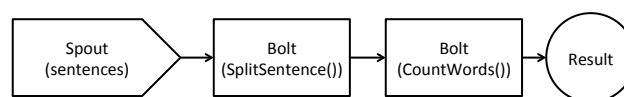


Figure 1: Graphical representation of the word count topology. This topology describes the stream that can be used to count words given an input of separate sentences.

Function	Description
<code>Bolt(FUNC,listen = 0,...)</code>	Used to create a new bolt. A bolt consists of an R function, and a specification of the bolt / spouts from which it receives tuples. The <code>listen</code> argument is used to indicate the order of bolts.
<code>Emit(x,...)</code>	Used to emit tuples from inside a bolt.
<code>RStorm(topology,...)</code>	Used to run a stream once a full topology has been specified.
<code>GetHash(name,...)</code>	Used to retrieve, inside a bolt, values from a hashmap.
<code>SetHash(name,data)</code>	Used to store, inside a bolt, values in a hashmap.
<code>Topology(spout,...)</code>	Used to create a topology by specifying the datasource (a <code>data.frame</code>) as the first spout.
<code>AddBolt(topology,bolt,...)</code>	Used to add a bolt to a stream. Once a bolt is added it receives an ID, which can be used in subsequent specification of bolts (<code>listen=ID</code>) to determine the order of the stream.
<code>Tuple(x,...)</code>	A single row <code>data.frame</code> . Used as the primary data format to be passed along the stream.

Table 1: Overview of the core functions and their primary parameters of **RStorm**.

a data stream consists of a *spout* – the data source – from which *tuples* are passed along a *topology*. The topology is a description of the spout and a series of *bolts*, which themselves are functional blocks of code. A bolt performs operations on *tuples*, the data objects that are passed between bolts in the stream. Bolts can store the results of their operations in a *local hashmap* (or database) and *emit* results (again tuples) to other bolts further down the topology. The topology, the bolts, the spout, the tuples, and the hashmap(s) together compose the most important concepts to understand a stream implemented in **RStorm**.

The *topology* is a description of the whole streaming process, and a solution to the word-count problem is given by the simple topology that is graphically presented in Figure 1. This topology describes that sentences (*tuples*) are emitted by the *spout*. These tuples – containing a full sentence – are analyzed by the first processing bolt. This first bolt, `SplitSentence(tuple)`, splits a sentence up into individual words and emits these single words as tuples. Next, these individual words are counted by the `CountWords(tuple)` bolt. The topology depicted in Figure 1 contains the core elements needed to understand the functioning of **RStorm** for a general streaming process. A topology consists of a description of the ordering of spouts and bolts in a stream. Tuples are the main data format to pass information between bolts. A call to `Emit(tuple,...)` within a bolt will make the emitted tuple available for other bolts. Table 1 summarizes the most important functions of the **RStorm** package to facilitate a stream and briefly explains their functionality.

Word count in RStorm and Java & Python

In **RStorm** the emulation of a streaming word count can be setup as follows: First, one loads **RStorm** and opens a datafile containing multiple sentences:

```
library(RStorm) # Include package RStorm
data(sentences)
```

The data, which is a `data.frame`, will function as the spout by emitting data from it row-by-row. After defining the spout, the functional bolts need to be specified. Table 2 presents both the **RStorm** as well as the Storm implementation of the first processing bolt. The Storm implementation is done partly in Java and partly in Python. For the **RStorm** implementation the full functional code is provided, while for the Storm implementation a number of details are omitted. However, it is easy to see how an

RStorm	Java
<pre># R function that receives a tuple # (a sentence in this case) # and splits it into words: SplitSentence <- function(tuple, ...) { # Split the sentence into words words <- unlist(strsplit(as.character(tuple\$sentence), " ")) # For each word emit a tuple for (word in words) Emit(Tuple(data.frame(word = word), ...)) }</pre>	<pre>/** * A Java function which makes * a call from the topology * to an external Python script: */ public SplitSentence() { super("Python", "splitsentence.py") } /* The Python script (.py) */ import storm class SplitSentenceBolt (storm.BasicBolt): def process(self, tuple) words = tuple.values[0].split(" ") for word in words: storm.emit([word])</pre>

Table 2: Description of the first functional bolt (`SplitSentence()`) of the word count stream in both **RStorm** (left), and Java (right).

actual Storm implementation maps to implementations in **RStorm**.

In both cases the `SplitSentence()` function receives tuples, each of which contains a sentence. Each sentence is split into words which are emitted further down the stream using the `Emit()` (or `storm.emit()`) function⁴. The second bolt is the `CountWords()` bolt, for which the **RStorm** code and the analogous Java code are presented in Table 3.

The `CountWords()` bolt receives tuples containing individual words. The **RStorm** implementation first uses the `GetHash()` function to get the entries of a hashmap / local-store called "wordcount". In production systems this often is a hashmap, or, if need be, some kind of database system. In **RStorm** this functionality is implemented using `GetHash` and `SetHash` as methods to easily store and retrieve objects. If the hashmap exists, the function subsequently checks whether the word is already in the hashmap. If the word is not found, the new word is added to the hashmap with a count of 1, otherwise the current count is incremented by 1.

After specifying the two bolts the *topology* needs to be specified. The topology determines the processing order of the streaming process. Table 4 presents how this is implemented in **RStorm** and Java⁵. Each time a bolt is added to a topology in **RStorm** the user is alerted to the position of that bolt within the stream, and the `listen` argument can be used to specify which emitted tuples a bolt should receive. Once the topology is fully specified, the stream can be run using the following call:

```
# Run the stream:
result <- RStorm(topology)
# Obtain results stored in "wordcount"
counts <- GetHash("wordcount", result)
```

The function `GetHash()` is overloaded for when the stream has finished and the function is used outside of a Bolt. It can be used to retrieve a hashmap once the result of a streaming process is passed to it as a second argument. The returned counts object is a `data.frame` containing columns of words and their associated counts and can be used to create a table of word counts.

The word count example shows the direct analogy between the implementation of a data stream in **RStorm** and in Storm. However, by focusing on an implementation that is analogous to the Storm implementation, a number of desirable R specific properties are lost. For example, the use of `for`

⁴Note that the `...` argument in the **RStorm** implementation is used to manage the stream and should thus always be supplied to the processing bolt.

⁵The `...` arguments in the Java implementation provide additional arguments used for managing parallelism in actual streaming applications.

RStorm	Java
<pre># R word counting function: CountWord <- function(tuple, ...) { # Get the hashmap "word count" words <- GetHash("wordcount") if (tuple\$word %in% words\$word) { # Increment the word count: words[words\$word == tuple\$word,] \$count <- words[words\$word == tuple\$word,] \$count + 1 } else { # If the word does not exist # Add the word with count 1 words <- rbind(words, data.frame(word = tuple\$word, count = 1)) } # Store the hashmap SetHash("wordcount", words) }</pre>	<pre>/** * A Java function which stores * the word count. */ public void execute(Tuple tuple, ...) { /* collect word from the tuple */ String word = tuple.getString(0); /* get counts from hashmap */ Integer count = counts.get(word); if (count == null) count = 0; /* increment counts */ count++; /* store counts */ counts.put(word, count); }</pre>

Table 3: Description of the second functional bolt (CountWord()) of the word count stream in both **RStorm** (left), and Java (right).

RStorm	Java
<pre># Setting up the R topology # Create topology: topology <- Topology(sentences) # Add the bolts: topology <- AddBolt(topology, Bolt(SplitSentence, listen = 0)) topology <- AddBolt(topology, Bolt(CountWord, listen = 1))</pre>	<pre>/** * Java core topology implementation */ /* Create topology */ TopologyBuilder builder = new TopologyBuilder(); /* Add the spout */ builder.setSpout("sentences", ...); /* Add the bolts */ builder.setBolt("split", new SplitSentence(), ... , .Grouping("sentences", ...) builder.setBolt("count", new WordCount(), ... , .Grouping("split"), ...)</pre>

Table 4: Specification of the topology using **RStorm** and Java. Note: The Java code is incomplete, but used only to illustrate the similarities between the two implementations.

(word in words) {...} in the word count example defies the efficient vectorisation of R, and thus is relatively slow. In R one would approach the word count problem (non streaming) differently: e.g., `table(unlist(strsplit(as.character(sentences$sentence), " "))`). The latter is much faster since it uses R properly, but the implementation in a data stream based on this code is not at all evident. Further note that while **RStorm** is modeled specifically after Storm, many other emergent streaming production packages – such as Yahoo!’s S4 – have a comparable structure. In all cases, the machinery to setup the stream can be separated from a number of functional pieces of code that update a set of parameters. These functional blocks of code are implemented in the **RStorm** bolts, and these can, after development in R, easily be implemented in production environments.

Sum of Squares method	Welford's method
<pre> var.SS <- function(x, ...) { # Get values stored in hashmap params <- GetHash("params1") if (!is.data.frame(params)) { # If no hashmap exists initialise: params <- list() params\$n <- params\$sum <- params\$sum2 <- 0 } # Perform updates: n <- params\$n + 1 S <- params\$sum + as.numeric(x[1]) SS <- params\$sum2 + as.numeric(x[1]^2) # Store the hashmap: SetHash("params1", data.frame(n = n, sum = S, sum2 = SS)) # Track the variance at time t: var <- 1/(n * (n-1)) * (n * SS - S^2) } TrackRow("var.SS", data.frame(var = var)) } </pre>	<pre> var.Welford <- function(x, ...) { x <- as.numeric(x[1]) params <- GetHash("params2") if (!is.data.frame(params)) { params <- list() params\$M <- params\$S <- params\$n <- 0 } n <- params\$n + 1 M <- params\$M + (x - params\$M) / (n + 1) S <- params\$S + (x - params\$M) * (x - M) SetHash("params2", data.frame(n = n, M = M, S = S)) var <- ifelse(n > 1, S / (n - 1), 0) TrackRow("var.Welford", data.frame(var = var)) } </pre>

Table 5: Comparison of two different bolts to compute a streaming variance.

RStorm examples

The following section shows a number of streaming examples and demonstrates some of **RStorm**'s additional features.

Example 1: Comparisons of streaming variance algorithms

This first example compares two bolts for the streaming computation of a sample variance. It introduces the `TrackRow(data)` functionality implemented in **RStorm** which can be used to monitor the progress of parameters at each time point in the stream. Table 5 shows two bolts with competing implementations of streaming variance algorithms. The first bolt uses the standard Sum of Squares algorithm, while the second uses Welford's method (Welford, 1962).

After specifying the functional bolts, the topology can be specified. Creating a topology object starts with the specification of a `data.frame`. This dataframe will be iterated through row-by-row to emulate a stream.

```

t <- 1000
x <- rnorm(t, 10^8, 1)
topology <- Topology(data.frame(x = x))

```

The spout defined in the object `topology` now contains a dataframe with a single column `x`, which contains 1000 draws from a Gaussian distribution with a large mean, $\mu = 10^8$, and a comparatively small variance, $\sigma^2 = 1$. Subsequently, the bolts are added to the topology:

```

topology <- AddBolt(topology, Bolt(var.SS, listen = 0))
topology <- AddBolt(topology, Bolt(var.Welford, listen = 0))
result <- RStorm(topology)

```

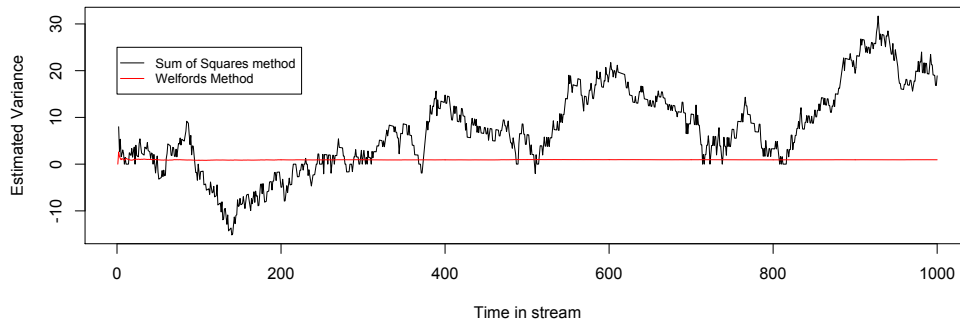


Figure 2: Comparison of two streaming variance algorithms. The sums of squares method (black) is numerically unstable when $\mu \gg \sigma^2$.

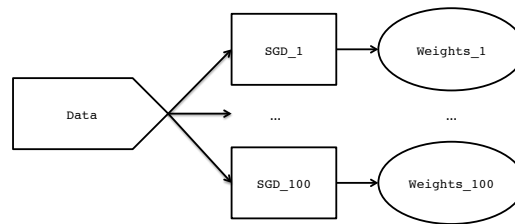


Figure 3: The DoNB SGD topology.

The `TrackRow()` function called within both functional bolts allows for inspection of the two variances at each point in time: using `TrackRow()` the values are stored for each time point. Using (e.g.) `GetTrack("var.SS", result)` on the result object after running the topology allows for the creation of Figure 2.

Example 2: Online gradient descent

This example provides an implementation in **RStorm** of an logistic regression using stochastic gradient descent (SGD; e.g., Zinkevich et al., 2010), together with a Double or Nothing (DoNB; Owen and Eckles, 2012) bootstrap to estimate the uncertainty of the parameters. The functional bolt first performs the sampling needed for the DoNB bootstrap and subsequently computes the update of the feature vector \tilde{w}_i :

```
StochasticGradientDescent <- function(tuple, learn = .5, boltID, ...) {
  if (rbinom(1, 1, .5) == 1) { # Only add the observation half of the times
    # get the set up weights for this bolt
    weights <- GetHash(paste("Weights_", boltID, sep = ""))
    if (!is.data.frame(weights)) {
      weights <- data.frame(beta = c(-1, 2))
    }
    w <- weights$beta # get weights-vector w
    y <- as.double(tuple[1]) # get scalar y
    X <- as.double(tuple[2:3]) # get feature-vector X
    grad <- (1 / (1 + exp(-t(w) %*% X)) - as.double(tuple[1])) * X
    SetHash(paste("Weights_", boltID, sep = ""),
             data.frame(beta = w - learn * grad)) # save weights
  } # otherwise ignore
}
```

The dataset for this example contains 1000 dichotomous outcomes using only a single predictor:

```
n <- 1000
X <- matrix(c(rep(1, n), rnorm(n, 0, 1)), ncol = 2)
beta <- c(1, 2)
y <- rbinom(n, 1, plogis(X %*% beta))
```

The DoNB is implemented by specifying *within* the functional bolt whether or not a datapoint in the stream should contribute to the update of the weights. Using the `boltID` parameter the same functional bolt can be used multiple times in the stream, each with its own local store. The topology is specified as follows:

```
topology <- Topology(data.frame(data), .verbose = FALSE)
for (i in 1:100) {
  topology <- AddBolt(topology, Bolt(StochasticGradientDescent,
    listen = 0, boltID = i), .verbose = FALSE)
}
```

This topology is represented graphically in Figure 3. After running the topology, the `GetHashList()` function can be used to retrieve all of the objects saved using `SetHash()` at once. This object is a list containing all the dataframes that are stored during the stream. It can be used to derive the estimates of β and the 95% confidence interval: $\beta_0 = 1.33$ [.50, 2.08] and $\beta_1 = 2.02$ [1.34, 2.76] which are close to the estimates obtained using `glm`: $\vec{\beta} = \{1.25, 2.04\}$.

Example 3: The k -arm bandit

The last example presents a situation in which streaming data naturally arises: bandit problems (e.g., Whittle, 1980). In the canonical bandit problem, the *two-armed Bernoulli bandit* problem, the data stream consists of rewards r_1, \dots, r_t which are observed after playing arm $a \in \{1, 2\}$ at time t' . The goal is to find a policy to decide between the two arms at $t = t'$ such that the cumulative reward $\mathcal{R} = \sum_{i=1}^t r_i$ is as large as possible.

RStorm can be used to compare competing solutions to the k -armed Bernoulli bandit problem. The data is composed of the reward r at time t for each of the actions a_1, \dots, a_k . The function below creates such a dataframe for usage in multiple simulation runs of different policies:

```
createCounterFactuals <- function(k = 2, t = 100, p.max = .5, epsilon = .1) {
  p <- c(p.max, rep(p.max - epsilon, k - 1))
  obs <- data.frame(matrix(rbinom(t * k, 1, p), ncol = k, byrow = TRUE))
}
```

This function creates a dataframe with k arms, where arm 1 has an expected payoff of `p.max`, and the other $k - 1$ arms have an expected payoff of `p.max - epsilon`. Here we compare playing the best action (optimal play – typically unknown) to a policy called Thompson sampling (Thompson, 1933; Scott, 2010). Each datapoint z_t emitted by the spout is a vector with the possible outcome of playing arm $1, \dots, k$ at time t .

For optimal play, the first bolt emits the reward observed by playing arm 1, and the second bolt uses a hashmap to compute the cumulative reward \mathcal{R}_{max} . The implementation of Thompson sampling, or Randomized Probability Matching (RPM, see Scott, 2010) uses three bolts: the first bolt (`selectRPM`) determines which arm to play given a set of estimates of the success for each arm and emits the observed reward. The second bolt (`updateRPM`) updates the estimated success of the arm that was played (using a simple beta-Bernoulli model), and the last bolt (`countRPM`) computes the cumulative reward \mathcal{R}_{rpm} . Both of the implementations are presented in Table 6.

The topology is graphically presented in Figure 4. The topology is initially specified using an empty dataset to enable the setup of multiple simulations:

```
topology <- Topology(data.frame())
topology <- AddBolt(topology, Bolt(selectMax, listen = 0))
topology <- AddBolt(topology, Bolt(countMax, listen = 1))
topology <- AddBolt(topology, Bolt(selectRPM, listen = 0))
topology <- AddBolt(topology, Bolt(updateRPM, listen = 3))
topology <- AddBolt(topology, Bolt(countRPM, listen = 3))
```

After specifying the bolts, the `ChangeSpout()` function is used to run the same topology with a different datasource. At each simulation run the spout is changed, and the regret, $\mathcal{R}_{max} - \mathcal{R}_{rpm}$, stored:

```
sims <- 100
regret <- rep(NA, sims)
for (i in 1:sims) {
  obs <- createCounterFactuals(k = 5, t = 10000, p.max = .5)
  topology <- ChangeSpout(topology, obs)
  result <- RStorm(topology)
  regret[i] <- GetHash("maxSum", result)$sum - GetHash("rpmSum", result)$sum
}
```

Play optimal	Play Thompson
<pre> # bolt which always selects arm 1: selectMax <- function(x, k.best = 1, ...) { # Always select the first arm # and emit: tuple <- Tuple(data.frame(best = x[,k.best])) Emit(tuple, ...) } # bolt which counts the rewards: countMax <- function(x, ...){ maxSum <- GetHash("maxSum") if (!is.data.frame(maxSum)) { maxSum <- data.frame(sum = 0) } sum <- maxSum\$sum + x\$best SetHash("maxSum", data.frame(sum = sum)) } </pre>	<pre> # bolt to select the action selectRPM <- function(x, ...) { arms <- length(x) rpmCoefs <- GetHash("coefs") # if no estimates set beta priors: if (!is.data.frame(rpmCoefs)) { rpmCoefs <- data.frame(arm = 1:arms, a = rep(1, arms), b = rep(1, arms)) SetHash("coefs", rpmCoefs) } # Get a random draw: draw <- daply(rpmCoefs, .(arm), .fun = function(x) return(rbeta(1, x\$a, x\$b))) # Determine which arm to play: rpm <- which.max(as.vector(draw)) tuple <- Tuple(data.frame(arm = rpm, rpm = x[,rpm])) Emit(tuple, ...) } # bolt to update the estimates updateRPM <- function(x, ...) { rpmCoefs <- GetHash("coefs") # update posteriors: rpmCoefs[x\$arm,]\$a <- rpmCoefs[x\$arm,]\$a + x\$rpm rpmCoefs[x\$arm,]\$b <- rpmCoefs[x\$arm,]\$b + (1 - x\$rpm) SetHash("coefs", rpmCoefs) } # bolt to count the reward countRPM <- function(x, ...) { # See "countMax()" for # implementation. # Values stored in hashmap "rpmSum" } </pre>

Table 6: Comparison of optimal play and Thompson sampling for the k-armed Bernoulli bandit problem.

After running 100 simulation runs with $p.\max = .5$ for $T = 10,000$ the average regret of Thompson sampling is 74.3, with an empirical 95% confidence interval of [43.9, 104.5].

Conclusions and limitations

Datasets in all areas of science are growing increasingly large, and they are often collected continuously. There is a need for novel analysis methods which synchronize current methodological advances with the emerging opportunities of streaming data. Streaming algorithms provide opportunities to deal with extremely large and ever growing data sets in (near) real time. However, the development of

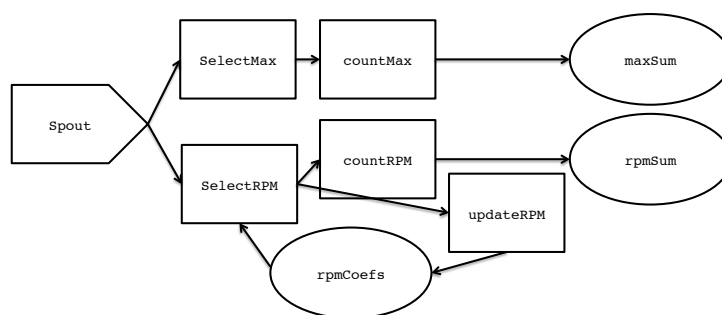


Figure 4: The k -armed bandit topology.

streaming algorithms for complex models is often cumbersome: the software packages that facilitate streaming processing in production environments do not provide statisticians with the simulation, estimation, and plotting tools they are used to. **RStorm** implements a streaming architecture modeled on Storm for easy development and testing of streaming algorithms in R.

In the future we intend to further develop the **RStorm** package to include a) default implementations of often occurring bolts (such as streaming means and variances of variables), and b) the ability to use, one-to-one, the bolts developed in **RStorm** in Storm. Storm provides the ability to write bolts in languages other than Java (for example Python, as demonstrated in the word count example). We hope to further develop **RStorm** such that true data streams in Storm can use functional bolts developed in R. **RStorm** is not designed as a scalable tool for production processing of data streams, and we do not believe that this is R's core strength. However, by providing the ability to test and develop functional bolts in R, and use these bolts directly in production streaming processing applications, **RStorm** aims to support users of R to quickly implement scalable and fault tolerant streaming applications.

Bibliography

- C. Anagnostopoulos, D. K. Tasoulis, N. M. Adams, N. G. Pavlidis, and D. J. Hand. Online linear and quadratic discriminant analysis with adaptive forgetting for streaming classification. *Statistical Analysis and Data Mining*, 5(2):139–166, 2012. [p123]
- B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems – PODS’02*, pages 1–16, New York, USA, 2002. ACM Press. [p123]
- L. Bottou. Online algorithms and stochastic approximations. In D. Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998. [p123]
- C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. *Advances in Neural Information Processing Systems*, 19(23):281–288, 2007. [p123]
- K. Gopalakrishna, F. Junqueira, M. Morel, L. Neumeyer, B. Robbins, and D. G. Ferro. S4, 2013. URL <http://incubator.apache.org/s4/team/>. [p123]
- M. Hahsler, M. Bolanos, and J. Forrest. *stream: Infrastructure for Data Stream Mining*, 2014. URL <http://CRAN.R-project.org/package=stream>. R package version 1.0-0. [p123]
- S. Michalak, A. DuBois, D. DuBois, S. V. Wiel, and J. Hogden. Developing systems for real-time streaming analysis. *Journal of Computational and Graphical Statistics*, 21(3):561–580, 2012. [p123]
- A. B. Owen and D. Eckles. Bootstrapping data arrays of arbitrary order. *The Annals of Applied Statistics*, 6(3):895–927, 2012. [p128]
- S. L. Scott. A modern Bayesian look at the multi-armed bandit. *Applied Stochastic Models in Business and Industry*, 26(6):639–658, 2010. [p129]
- Storm User Group. Storm. Distributed and fault-tolerant realtime computations, 2013. URL <http://storm-project.net>. [p123]
- W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3–4):285–294, 1933. [p129]

- B. P. Welford. Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4(3):419–420, 1962. [p127]
- P. Whittle. Multi-armed bandits and the Gittins index. *Journal of the Royal Statistical Society B*, 42(2): 143–149, 1980. [p129]
- M. A. Zinkevich, A. Smola, and M. Weimer. Parallelized stochastic gradient descent. *Advances in Neural Information Processing Systems*, 23(6):1–9, 2010. [p128]

Maurits Kaptein
Department of Methodology and Statistics
Tilburg University, Tilburg, the Netherlands
Archipelstraat 13
6524LK Nijmegen, the Netherlands
+31 6 21262211
maurits@mauritskaptein.com

The gridSVG Package

by Paul Murrell and Simon Potter

Abstract The **gridSVG** package can be used to generate a **grid**-based R plot in an SVG format, with the ability to add special effects to the plot. The special effects include animation, interactivity, and advanced graphical features, such as masks and filters. This article provides a basic introduction to important functions in the **gridSVG** package and discusses the advantages and disadvantages of **gridSVG** compared to similar R packages.

Introduction

The SVG graphics format (Dengler et al., 2011) is a good format for including plots in web pages because it is a vector format (so it scales well) and because it offers features for animation and interactivity. SVG also integrates well with other important web technologies such as HTML and JavaScript.

It is possible to produce a static R plot in an SVG format with the built-in `svg()` function (from the **grDevices** package), but the **gridSVG** package (Murrell and Potter) provides an alternative way to generate an SVG plot that allows for creating animated and interactive graphics.

There are two types of graphics functions in R: functions based on the default **graphics** package and functions based on the **grid** graphics package. As the package name suggests, the **gridSVG** package only works with a plot that is drawn using the **grid** graphics package. This includes plots from several important graphics packages in R, such as **lattice** (Sarkar, 2008) and **ggplot2** (Wickham, 2009), but **gridSVG** does not work with all plots that can be produced in R.

This article demonstrates basic usage of the **gridSVG** package and outlines some of the ways that **gridSVG** can be used to produce graphical results that are not possible in standard R graphics. There is also a discussion of other packages that provide ways to generate dynamic and interactive graphics for the web and the strengths and weaknesses of **gridSVG** compared to those packages.

Basic usage

The following code draws a **lattice** multi-panel plot (see Figure 1).

```
> library(lattice)

> dotplot(variety ~ yield | site, data = barley, groups = year,
          key = simpleKey(levels(barley$year), space = "right"),
          subset = as.numeric(site) < 4, layout = c(1, 3))
```

The `grid.export()` function in **gridSVG** converts the current (**grid**) scene on the active graphics device to an SVG format in an external file.

```
> library(gridSVG)

> grid.export("lattice.svg")
```

This SVG file can be viewed directly in a browser (see Figure 2) or embedded within HTML as part of a larger web page.

This usage of **gridSVG**, to produce a static SVG version of an R plot for use on the web, offers no obvious benefit compared to the built-in `svg()` graphics device. However, the **gridSVG** package provides several other functions that can be used to enhance the SVG version of an R plot.

A simple example

In order to demonstrate, with code, some of the distinctive features of **gridSVG**, we introduce a simple **grid** scene that is inspired by the Monty Hall problem.¹

```
> library(grid)
```

¹http://en.wikipedia.org/wiki/Monty_Hall_problem

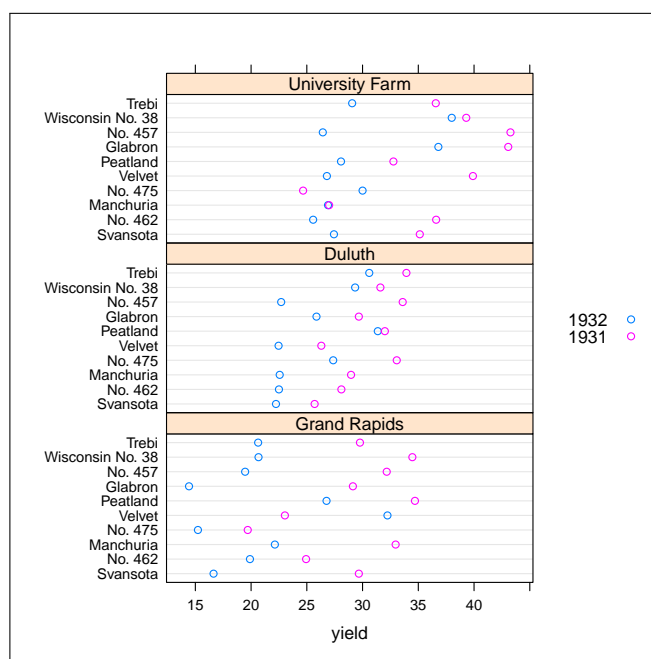


Figure 1: A `lattice` multi-panel plot drawn on the standard `pdf()` graphics device.

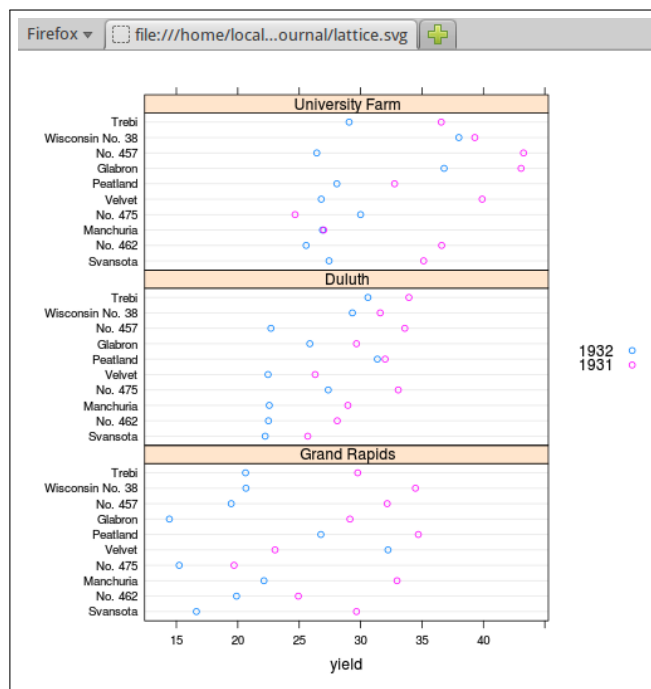


Figure 2: The `lattice` plot from Figure 1 exported to an SVG file by `gridSVG` and viewed in Firefox. This demonstrates that a static R plot can be converted to an SVG format with `gridSVG` for use on the web.

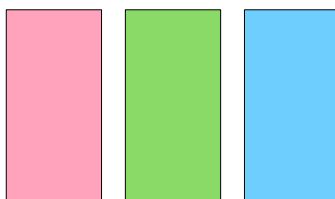


Figure 3: A diagram of the Monty Hall problem, drawn using **grid**. Hidden behind each rectangle is either the word “goat” or the word “car”).

The scene consists of three words, “goat”, “goat”, and “car”, drawn in random order across the page, with an opaque rectangle drawn on top of each word.

In relation to the Monty Hall problem, the three rectangles represent three “doors”, behind which are hidden two goats and a car. A “contestant” must choose a door and then he or she gets the “prize” behind that door. However, after the contestant has chosen a door, a “game show host” opens one of the other doors to reveal a “goat” and the contestant gets the opportunity to change to the remaining unopened door or stick with the original choice. Should the contestant stick or switch?²

The following code produces the scene and the result is shown in Figure 3. The main drawing code is wrapped up in a function so that we can reuse it later on.

```
> text <- sample(c("goat", "goat", "car"))
> cols <- hcl(c(0, 120, 240), 80, 80)

> MontyHall <- function() {
  grid.newpage()
  grid.text(text, 1:3/4, gp = gpar(cex = 2), name = "prizes")
  for (i in 1:3) {
    grid.rect(i/4 - .1, width=.2, height=.8, just = "left",
              gp = gpar(fill = cols[i]), name = paste0("door", i))
  }
}

> MontyHall()
```

The code in the `MontyHall()` function makes use of the fact that **grid** functions allow names to be associated with the objects in a scene. In this case, the three rectangles in this scene have been given names—“door1”, “door2”, and “door3”—and the text has been given the name “prizes”.

The **grid** function `grid.ls()` can be used to display the names of all objects in a scene.

```
> grid.ls(fullNames = TRUE)

text[prizes]
rect[door1]
rect[door2]
rect[door3]
```

These names will be used later to identify the rectangles so that we can modify them to generate special effects.

Hyperlinks

The `grid.hyperlink()` function from the **gridSVG** package can be used to add hyperlinks to parts of a **grid** scene. For example, the following code adds a link to each door so that clicking on a door (while viewing the SVG version of the scene in a browser) leads to a Google Image Search on either “car” or “goat” depending on what is behind the door. The first argument to `grid.hyperlink()` is the name of the **grid** object with which to associate the hyperlink. The `href` argument provides the actual link and the `show` argument specifies how to show the target of the link (“new” means open a new tab or window).

²An exercise for the reader is to determine which door conceals the car based on the R code and figures presented in this article.

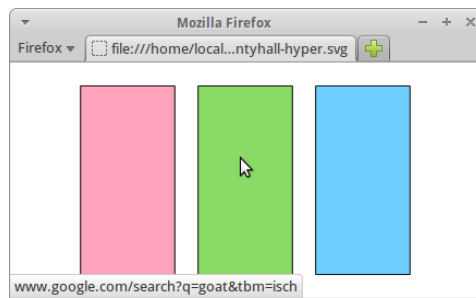


Figure 4: The Monty Hall image, with a hyperlink on each door. The mouse is hovering over the middle door and the browser is showing the hyperlink target in the bottom-left of its window. If we click the mouse, we will navigate to a Google Image Search for the word “goat”.

```
> library(gridSVG)

> links <- c("http://www.google.com/search?q=car&tbm=isch",
            "http://www.google.com/search?q=goat&tbm=isch")

> for (i in 1:3) {
  grid.hyperlink(paste0("door", i),
                href = links[match(text[i], c("car", "goat"))],
                show = "new")
}
```

After running this code, the scene is completely unchanged on a normal graphics device, but if we use `grid.export()` to convert the scene to SVG, we end up with an image that contains hyperlinks. Figure 4 shows the result, with the mouse hovering over the middle door; at the bottom-left of the browser window, we can see from the hyperlink that there is a goat behind this door.

```
> grid.export("montyhall-hyper.svg")
```

Animation

The function `grid.animate()` from **gridSVG** allows us to animate the features of shapes in a **grid** scene. For example, the following code draws the Monty Hall scene again and then animates the width of the middle door so that it slides open (to reveal the word “goat”). The first argument to `grid.animate()` is the name of the object to animate. Subsequent arguments specify which feature of the object to animate, in this case width, plus the values for the animation. The duration argument controls how long the animation will last.

```
> MontyHall()
> goatDoor <- grep("goat", text)[1]
> grid.animate(paste0("door", goatDoor), width = c(.2, 0), duration = 2)

> grid.export("montyhall-anim.svg")
```

Again, no change is visible on a normal R graphics device, but if we export to SVG and view the result in a browser, we see the animation (see Figure 5).

Advanced graphics features

The **gridSVG** package offers several graphics features that are not available in standard R graphics devices. These include non-rectangular clipping paths, masks, fill patterns and fill gradients, and filters (Murrell and Potter, 2013). This section demonstrates the use of a mask on the Monty Hall scene.

A mask is a greyscale image that is used to affect the transparency (or alpha-channel) of another image: anywhere the mask is white, the masked image is fully visible; anywhere the mask is black, the masked image is invisible; and anywhere the mask is grey, the masked image is semitransparent.

The following code uses standard **grid** functions to define a simple scene consisting of a white cross on top of a grey circle on a white background, which we will use as a mask (see Figure 6). Any **grid** scene can be used to create a mask; in this case, we use the `gTree()` function from **grid** to create a graphical object that is a collection of several other graphical objects.

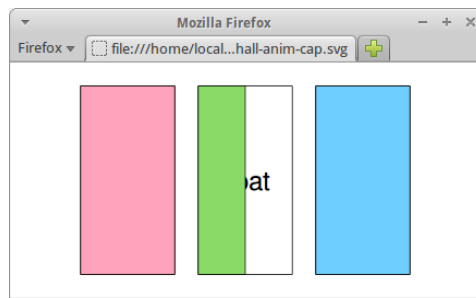


Figure 5: The Monty Hall image, with the middle “door” animated so that it slides open (to reveal the word “goat”).



Figure 6: Using a mask on an image. The picture on the left shows a white cross on top of a grey circle on a white background. This is used as a mask on the rectangles in the Monty Hall image on the right. The effect is to create a semitransparent window in the middle door (through which we can glimpse the word “goat”).

```
> circleMask <- gTree(children = gList(rectGrob(gp = gpar(col = NA, fill = "white")),
  circleGrob(x = goatDoor/4, r=.15,
    gp = gpar(col = NA, fill = "grey")),
  polylineGrob(c(0, 1, .5, .5),
    c(.5, .5, 0, 1),
    id = rep(1:2, each = 2),
    gp = gpar(lwd = 10, col = "white"))))
```

The next code shows how this crossed circle on a white background can be used as a mask to affect the transparency of one of the rectangles in the Monty Hall scene. The functions `mask()` and `grid.mask()` are from **gridSVG**. The `mask()` function takes a **grid** object (as generated above) and turns it into a mask object. The `grid.mask()` function takes the name of a **grid** object to mask, plus the mask object produced by `mask()`.

```
> MontyHall()
> grid.mask(paste0("door", goatDoor), mask(circleMask))
> grid.export("montyhall-masked.svg")
```

The effect of the mask is shown in Figure 6.

Interactivity

The `grid.garnish()` function in the **gridSVG** package opens up a broad range of possibilities for enhancing a **grid** scene, particularly for adding interactivity to the scene.

A simple example is shown in the code below. Here we are adding tooltips to each of the doors in the Monty Hall scene so that hovering the mouse over a door produces a label that shows what is behind the door (see Figure 7). The first argument to `grid.garnish()` is the name of the object to modify. Subsequent arguments specify SVG attributes to add to the object; in this case, we add a title attribute, which results in a tooltip (in some browsers).



Figure 7: The Monty Hall image with tooltips added to each door. The mouse is hovering over the middle door, which results in a tooltip being displayed to show that there is a “goat” behind this door.

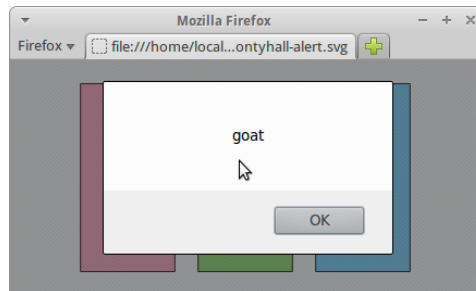


Figure 8: The Monty Hall image with interactivity. The mouse has just been clicked on the middle door, which has resulted in an alert box popping up to show that this door has a “goat” behind it.

```
> MontyHall()
> for (i in 1:3) {
  grid.garnish(paste0("door", i), title = text[i])
}
> grid.export("montyhall-tooltip.svg")
```

The `grid.garnish()` function can also be used to associate JavaScript code with an object in the scene. The following code shows a simple example where clicking on one of the rectangles pops up an alert box showing what is behind that door (see Figure 8). The attribute in this example is `onclick`, which is used to define an action that occurs when the object is clicked with the mouse (in a browser).

```
> MontyHall()
> for (i in 1:3) {
  grid.garnish(paste0("door", i),
    onclick = paste0("alert('", text[i], "')"))
}
> grid.export("montyhall-alert.svg")
```

For more complex interactions, it is possible to include JavaScript code within the scene, using the `grid.script()` function, so that an event on an object within the scene can be associated with a JavaScript function call to perform a more sophisticated action. The code below shows a simple example where clicking on one of the rectangles in the Monty Hall scene will call the JavaScript function `open()` to “open” the door (by making the rectangle invisible; see Figure 9). The `open()` function is defined in a separate file called “MontyHall.js” (shown in Figure 10).

```
> MontyHall()
> for (i in 1:3) {
  grid.garnish(paste0("door", i), onclick = "open(evt)")
}
> grid.script(file = "MontyHall.js")
> grid.export("montyhall-js.svg")
```

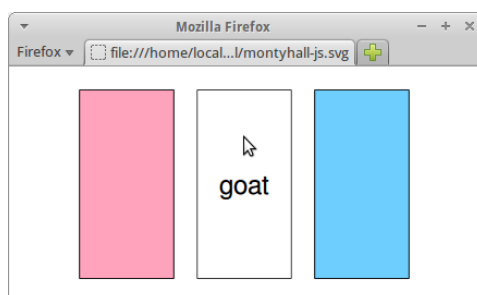



Figure 9: The Monty Hall image with more interactivity. The mouse has just been clicked on the middle door, which has resulted in the middle door becoming invisible, thereby revealing a “goat” behind the door.

```
open = function(e) {
  e.currentTarget.setAttribute("visibility", "hidden");
}
```

Figure 10: The JavaScript code used in Figure 9 that defines the `open()` function to “open” a door by making the rectangle invisible.

A more complex demonstration

The previous section kept things very simple in order to explain the main features of **gridSVG**. In this section, we present a more complex example which involves adding interactivity to a **lattice** multi-panel plot. The following code generates the **lattice** plot from Figure 1.

```
> dotplot(variety ~ yield | site, data = barley, groups = year,
  key = simpleKey(levels(barley$year), space = "right"),
  subset = as.numeric(site) < 4, layout = c(1, 3))
```

Because the **lattice** package is built on **grid**, and because the **lattice** package names all of the objects that it draws,³ there are names for every object drawn in this plot. The following code uses the **grid** function `grid.grep()`⁴ to show some of the named objects in this plot (in this case, all objects that have a name that contains “xyplot.points”). These are the objects that represent the data symbols within the **lattice** plot.

```
> grid.grep("xyplot.points", grep = TRUE, global = TRUE)
```

```
[[1]]
plot_01.xyplot.points.group.1.panel.1.1
```

```
[[2]]
plot_01.xyplot.points.group.2.panel.1.1
```

```
[[3]]
plot_01.xyplot.points.group.1.panel.1.2
```

```
[[4]]
plot_01.xyplot.points.group.2.panel.1.2
```

```
[[5]]
plot_01.xyplot.points.group.1.panel.1.3
```

```
[[6]]
plot_01.xyplot.points.group.2.panel.1.3
```

The following code uses `grid.garnish()` to add event handlers to the objects that represent the points in the plot, so that JavaScript functions are called whenever the mouse moves over a point and whenever the mouse moves off the point again.

³<http://lattice.r-forge.r-project.org/Vignettes/src/naming-scheme/namingScheme.pdf>

⁴Introduced in R version 3.1.0.

```

function highlight(evt) {
  var element = evt.currentTarget;
  var id = element.id;
  var index = id.substring(id.search(/[0-9]+$/)) + 1, id.length);
  for (var panel=1;panel<4;panel++) {
    for (var group=1;group<3;group++) {
      var selid =
        'plot_01.xyplot.points.group.'+group+'.panel.1.'+panel+'.1.'+index;
      var dot = document.getElementById(selid);
      dot.setAttribute("stroke-width", "6");
    }
  }
}

function unhighlight(evt) {
  var element = evt.currentTarget;
  var id = element.id;
  var index = id.substring(id.search(/[0-9]+$/)) + 1, id.length);
  for (var panel=1;panel<4;panel++) {
    for (var group=1;group<3;group++) {
      var selid =
        'plot_01.xyplot.points.group.'+group+'.panel.1.'+panel+'.1.'+index;
      var dot = document.getElementById(selid);
      dot.setAttribute("stroke-width", "1");
    }
  }
}

```

Figure 11: The JavaScript code that defines the `highlight()` and `unhighlight()` functions to implement linked selection of points for the **lattice** plot in Figure 12.

```

> numPoints <- length(levels(barley$variety))
> grid.garnish("xyplot.points", grep = TRUE, global = TRUE, group = FALSE,
  onmouseover = rep("highlight(evt)", numPoints),
  onmouseout = rep("unhighlight(evt)", numPoints),
  "pointer-events" = rep("all", numPoints))

```

This use of `grid.garnish()` differs from the previous simple examples because it has an effect on several **grid** objects, rather than just one. The `grep` and `global` arguments specify that the name, "xyplot.points", should be treated as a regular expression and the garnish will affect all objects in the scene with a name that matches that pattern. Furthermore, each **grid** object that matches represents several data symbols, so the `group` argument is used to specify that the garnish should be applied to each individual data symbol. Because, for each object, the garnish is being applied to multiple data symbols, we must provide multiple values, which explains the use of `rep()` for the `onmouseover`, `onmouseout`, and `pointer-events` arguments.

The JavaScript code that defines the event handlers `highlight()` and `unhighlight()` is shown in Figure 11. A detailed explanation of this code is beyond the scope of this article, but it should be clear that these functions are relatively simple, just looping over the two groups in each panel, and over the three panels, to highlight (or unhighlight) all points that share the same index.

This JavaScript code is added to the plot using `grid.garnish()`, and then the whole scene is exported to SVG with `grid.export()`.

```

> grid.script(file = "lattice-brush.js")
> grid.export("lattice-brush.svg")

```

A snapshot of the final result is shown in Figure 12, with the mouse over one point and all related points highlighted.

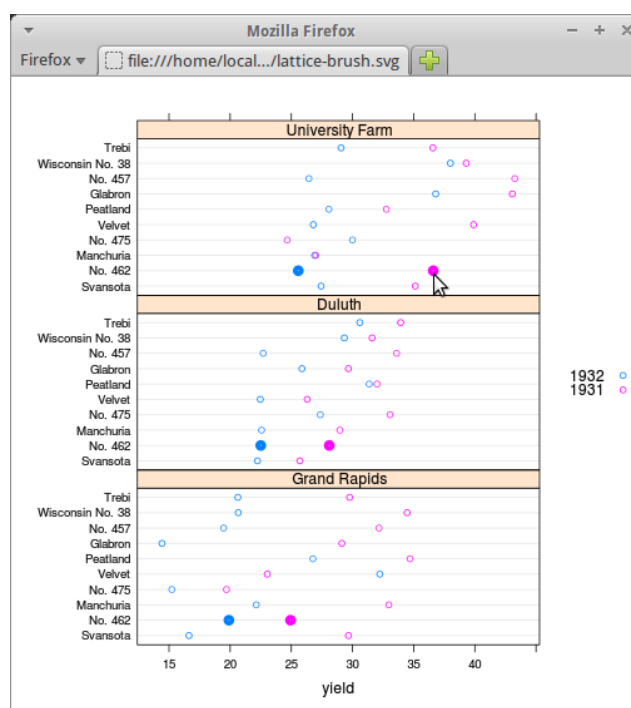


Figure 12: The **lattice** plot from Figure 1 with interaction added. Moving the mouse over a point highlights the point and highlights every other “related” point in all panels of the plot.

Limitations

The **gridSVG** package provides an opportunity to produce more sophisticated, more dynamic, and more interactive R plots compared to the standard R graphics devices. However, there are some strict limitations on what can be achieved with this package.

First of all, the package only works for plots that are based on the **grid** graphics system. This includes some major graphics packages, such as **lattice** and **ggplot2**, but excludes a large amount of graphics functionality that is only available in the default **graphics** package or packages that build on **graphics**. Given a plot from a function that is not based on **grid**, the **gridSVG** package will only produce a blank SVG file.

Another limitation is that **gridSVG** does not generate any JavaScript code itself. This means that anything beyond the most basic interactivity will require the user to write JavaScript code, which imposes a burden on the user in terms of both time and knowledge.

Another point that has only briefly been acknowledged in the example R code so far is that the **gridSVG** functions that add special features to a **grid** scene (such as hyperlinks and animation) rely heavily on the ability to *identify* specific components of a **grid** scene. The Monty Hall examples all rely on the fact that the rectangles that are drawn to represent doors each have a name—“door1”, “door2”, and “door3”—and the code that adds hyperlinks or animation identifies the rectangles by using these names. This means that **gridSVG** is dependent upon an appropriate naming scheme being used for any **grid** drawing (Murrell, 2012). This requirement is met by the **lattice** package and, to a lesser extent by the **ggplot2** package, but cannot be relied on in general.

Alternative approaches

The **gridSVG** package provides one way to produce dynamic and interactive versions of R plots for use on the web, but there are several other packages that provide alternative routes to the same destination. This section discusses the differences between **gridSVG** and several other packages that have similar goals.

The **animation** package (Xie, 2013) provides a convenient front-end for producing animations in various formats (some of which are appropriate for use on the web), but the approach is frame-based (draw lots of separate images and then stitch them together to make an animation). The advantage of an SVG-based approach to animation is that the animation is declarative, which means that the

animation can be described more succinctly and efficiently and the resulting animation will often appear smoother. On the other hand, the **animation** package will work with any R graphics output; it is not restricted to just **grid**-based output.

The **SVGAnnotation** package (Nolan and Temple Lang, 2012) performs a very similar role to **gridSVG**, by providing functions to export R plots to an SVG format with the possibility of adding dynamic and interactive features. One major advantage of **SVGAnnotation** is that it will export R plots that are based on the standard **graphics** package (as well as plots that are based on **grid**). **SVGAnnotation** also provides some higher-level functions that automatically generate JavaScript code to implement specific sorts of more complex interactivity. For example, the `linkPlots()` function can be used to generate linked plots, where moving the mouse over a data symbol in one plot automatically highlights a corresponding point in another plot. The main disadvantage of **SVGAnnotation** is that it works with the SVG that is produced by the built-in `svg()` device, which is much less structured than the SVG that is generated by **gridSVG**. That is not a problem if the functions that **SVGAnnotation** provides do everything that we need, but it makes for much more work if we need to, for example, write our own JavaScript code to work with the SVG that **SVGAnnotation** has generated.

Another package that can export R graphics output to SVG is the **RSVGTipsDevice** package (Plate, 2011). This package creates a standard R graphics device, so it can export any R graphics output, but it is limited to adding tooltips and hyperlinks. This package also requires the tooltips or hyperlinks to be added at the time that the R graphics output is produced, rather than after-the-fact using names to refer to previously-drawn output. This makes it harder to associate tooltips or hyperlinks with output that is produced by someone else's code, such as a complex **lattice** plot.

A number of packages, including **rCharts** and **googleVis** (Vaidyanathan, 2013; Gesmann and de Castillo, 2011), provide a quite different approach to producing dynamic and interactive plots for the web. These packages outsource the plot drawing to JavaScript libraries such as NVD3, highcharts, and the Google Visualisation API (Novus, 2012; Highsoft AS, 2013; Google, 2013). The difference here is that the plots produced are not R plots. The advantage is that very little R code is required to produce a nice result, provided the JavaScript library can produce the style of plot and the sort of interactivity that we want.

Another approach to interactivity that is implemented in several packages, notably **shiny** (RStudio Inc., 2013), involves running R as a web server and producing new R graphics in response to user events in the browser. The difference here is that the user typically interacts with GUI widgets (buttons and menus) outside the graphic and each user event generates a completely new R graphic. With **gridSVG**, the user can interact directly with elements of the graphic itself and all of the changes to the graphic occur in the browser with no further need of R.

In summary, using the **gridSVG** package is appropriate if we want to add advanced graphics features to a **grid**-based R plot, or if we want to add dynamic or interactive elements to a **grid**-based R plot, particularly if we want to produce a result that is not already provided by a high-level function in the **SVGAnnotation** package. An approach that holds some promise is to generate SVG content using **gridSVG** and then manipulate that content by adding JavaScript code based on a sophisticated JavaScript library such as d3 (Bostock et al., 2011) and Snap.svg (Baranovskiy, 2014).

Availability

The **gridSVG** package is available from CRAN. The code examples in this article are known to work for **gridSVG** versions 1.3 and 1.4 using Firefox 28.0 on Ubuntu 12.04.

Support for SVG varies between browsers, for example Chrome 34.0 on Ubuntu 12.04 does not produce the tooltips in Figure 7. Several web sites provide summary tables of supported SVG features.⁵ Differences between browser JavaScript engines is another potential source of variation. Nevertheless, all major browsers now provide native support of at least basic SVG features, several mature and stable JavaScript libraries are available to abstract away browser differences, and the situation is constantly improving.

Online versions of the figures in this article are available from <http://www.stat.auckland.ac.nz/~paul/Reports/gridSVGrjV2/>. Further documentation and examples for **gridSVG** are available from <https://www.stat.auckland.ac.nz/~paul/R/gridSVG/>.

⁵<http://caniuse.com/svg>
http://en.wikipedia.org/wiki/Comparison_of_layout_engines_%28Scalable_Vector_Graphics%29

Acknowledgements

We would like to thank the anonymous reviewers for many helpful comments that lead to improvements in this article.

Bibliography

- D. Baranovskiy. Snap.svg, 2014. URL <http://snapsvg.io/>. [p142]
- M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-Driven Documents. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2011. URL <http://vis.stanford.edu/papers/d3>. [p142]
- P. Dengler, D. Jackson, C. Lilley, J. Fujisawa, C. McCormack, E. Dahlström, A. Grasso, J. Ferraiolo, D. Schepers, and J. Watt. Scalable vector graphics (SVG) 1.1 (second edition). W3C recommendation, W3C, Aug. 2011. <http://www.w3.org/TR/2011/REC-SVG11-20110816/>. [p133]
- M. Gesmann and D. de Castillo. googleVis: Interface between R and the Google Visualisation API. *The R Journal*, 3(2):40–44, December 2011. URL http://journal.r-project.org/archive/2011-2/RJournal_2011-2_Gesmann+de~Castillo.pdf. [p142]
- Google. Google Visualization API, 2013. URL <https://developers.google.com/chart/interactive/docs/reference>. [p142]
- Highsoft AS. Highcharts JS, 2013. URL <http://www.highcharts.com/>. [p142]
- P. Murrell. What’s in a Name? *The R Journal*, 4(2):5–12, dec 2012. URL http://journal.r-project.org/archive/2012-2/RJournal_2012-2_Murrell.pdf. [p141]
- P. Murrell and S. Potter. *gridSVG: Export grid graphics as SVG*. R package version 1.4-0. [p133]
- P. Murrell and S. Potter. Advanced SVG Graphics from R. Technical Report 2013-7, Department of Statistics, The University of Auckland, 2013. URL <http://stattech.wordpress.fos.auckland.ac.nz/2013-7-advanced-svg-graphics-from-r/>. [p136]
- D. Nolan and D. Temple Lang. Interactive and animated scalable vector graphics and R data displays. *Journal of Statistical Software*, 46(1):1–88, 1 2012. ISSN 1548-7660. URL <http://www.jstatsoft.org/v46/i01>. [p142]
- Novus. NVD3.js : Re-usable charts for d3.js, 2012. URL <http://nvd3.org/>. [p142]
- T. Plate. *RSVGTipsDevice: An R SVG graphics device with dynamic tips and hyperlinks*, 2011. URL <http://CRAN.R-project.org/package=RSVGTipsDevice>. R package version 1.0-4. [p142]
- RStudio Inc. *shiny: Web Application Framework for R*, 2013. URL <http://CRAN.R-project.org/package=shiny>. R package version 0.3.0. [p142]
- D. Sarkar. *Lattice: Multivariate Data Visualization with R*. Springer-Verlag, New York, 2008. URL <http://lmdvr.r-forge.r-project.org>. ISBN 978-0-387-75968-5. [p133]
- R. Vaidyanathan. *rCharts: Interactive Charts using Polycharts.js*, 2013. R package version 0.4.2. [p142]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag, New York, 2009. ISBN 978-0-387-98140-6. URL <http://had.co.nz/ggplot2/book>. [p133]
- Y. Xie. animation: An R package for creating animations and demonstrating statistical methods. *Journal of Statistical Software*, 53:1–27, 2013. URL <http://www.jstatsoft.org/v53/i01/>. [p141]

Paul Murrell
The University of Auckland
Auckland
New Zealand paul@stat.auckland.ac.nz

Simon Potter
The University of Auckland
Auckland
New Zealand simon@sjp.co.nz

MRCV: A Package for Analyzing Categorical Variables with Multiple Response Options

by Natalie A. Koziol and Christopher R. Bilder

Abstract Multiple response categorical variables (MRCVs), also known as “pick any” or “choose all that apply” variables, summarize survey questions for which respondents are allowed to select more than one category response option. Traditional methods for analyzing the association between categorical variables are not appropriate with MRCVs due to the within-subject dependence among responses. We have developed the **MRCV** package as the first R package available to correctly analyze MRCV data. Statistical methods offered by our package include counterparts to traditional Pearson chi-square tests for independence and loglinear models, where bootstrap methods and Rao-Scott adjustments are relied on to obtain valid inferences. We demonstrate the primary functions within the package by analyzing data from a survey assessing the swine waste management practices of Kansas farmers.

Introduction

Survey questions often instruct respondents to “choose all that apply” from a list of response categories. For example, surveys instituted by U.S. government agencies are mandated to ask race and ethnicity questions in this format ([Office of Management and Budget, 1997](#), p. 58781). In medical applications, “choose all that apply” questions have been used for a variety of purposes, including gathering information about treatment and monitoring strategies ([Kantarjian et al., 2007](#); [Riegel et al., 2006](#)). Outside of surveys, this format can appear in unexpected applications. For example, wildlife management researchers are often interested in the food habits of animal species. Traces of prey in scats provide these researchers with a “choose all that apply” type of response because multiple prey types may be present ([Lemons et al., 2010](#); [Riemer et al., 2011](#)).

Variables that summarize data arising from a “choose all that apply” format are referred to as multiple response categorical variables (MRCVs), and the response categories within each MRCV are referred to as items ([Bilder and Loughin, 2004](#)). Because individual subjects are allowed to choose multiple items, the responses are likely dependent, and therefore traditional methods for analyzing categorical variables (e.g., Pearson chi-square tests for independence, loglinear models) are not appropriate. Unfortunately, numerous examples exist where these traditional methods are still used (see [Wright, 2010](#) for a review), which can lead to erroneous results ([Loughin and Scherer, 1998](#)).

While MRCVs have been identified since at least [Coombs \(1964\)](#), methods for correctly analyzing MRCVs in the context of common categorical data analysis interests, such as examining associations between variables, have only been available for approximately 15 years (e.g., see [Agresti and Liu, 1999](#)). Our **MRCV** package ([Koziol and Bilder, 2014](#)) is the first R package available to implement valid inference procedures for this type of data. The functions within the package can be used by researchers who want to examine the relationship among items from up to three MRCVs.

We begin this paper by first illustrating functions within the package for summarizing MRCV data and testing for independence. Then, we illustrate functions for fitting a generalized loglinear model to MRCV data and for performing follow-up analyses using method functions. Our examples focus on only two MRCVs for brevity reasons, but we discuss extensions in the conclusion.

Test for independence

We begin with an example from [Bilder and Loughin \(2007\)](#) involving a simple random sample of Kansas swine farmers. There are two MRCVs to be examined here, and we denote them generically as W and Y . The first MRCV (W) corresponds to a survey question that asked farmers to state which contaminants they tested for from the items “nitrogen”, “phosphorous”, and “salt” (W_1 , W_2 , W_3 , respectively). The second MRCV (Y) corresponds to a survey question that asked farmers to identify their swine waste storage methods from the items “lagoon”, “pit”, “natural drainage”, and “holding tank” (Y_1 , Y_2 , Y_3 , Y_4 , respectively). Farmers were instructed to “choose all that apply” from each of these predefined lists. By using a 0 to denote an item not chosen (negative response) and a 1 to denote an item chosen (positive response), each observation consists of a set of correlated binary responses, as shown below:


```
> head(farmer2, n = 3)
      w1 w2 w3 y1 y2 y3 y4
1  0  0  0  0  0  0  0
2  0  0  0  0  0  0  1
3  0  0  0  0  0  0  1

> tail(farmer2, n = 3)
      w1 w2 w3 y1 y2 y3 y4
277  1  1  1  1  1  0  0
278  1  1  1  1  1  0  0
279  1  1  1  1  1  1  0
```

We see, for example, that the third farmer does not test for any contaminants and uses only a holding tank for waste storage.

Contingency table-like summaries of MRCV data are often given in papers. In particular, marginal counts for all pairwise positive responses between items in W and Y are shown in Table 1. This display format can lead researchers to want to apply Pearson chi-square tests (or other simple categorical measures) to the table of counts in order to understand associations between the MRCVs. However, this approach is not correct because it does not take into account the fact that an individual subject can contribute to multiple counts in the table, which violates any type of multinomial distribution underlying assumption for these specific counts. Furthermore, three other tables summarizing the pairwise positive/negative responses (e.g., summarizing responses for items “not” chosen) of this type could also be constructed. Agresti and Liu (1999) and Bilder and Loughin (2001) show that testing procedures are not invariant to whether positive or negative responses are summarized and that different conclusions about the data can be reached depending on the types of responses summarized.

Examining all possible combinations of the positive/negative item responses between MRCVs is the preferred way to display and subsequently analyze MRCV data. The `item.response.table()` function provides this summary for each (W_i, Y_j) pair:

```
> item.response.table(data = farmer2, I = 3, J = 4)

      y1      y2      y3      y4
      0  1      0  1      0  1      0  1
w1 0 123 116 175 64 156 83 228 11
   1  13  27  24 16  38  2  38  2
w2 0 128 121 181 68 165 84 237 12
   1   8  22  18 12  29  1  29  1
w3 0 134 124 184 74 174 84 245 13
   1   2  19  15  6  20  1  21  0
```

where I is the number of items for W and J is the number of items for Y . The pairwise item-response table indicates, for example, that 27 farmers tested for nitrogen and used lagoon as a waste storage method (i.e., $W_1 = 1, Y_1 = 1$). Furthermore, 123 farmers did not test for nitrogen and did not use a lagoon, 13 farmers tested for nitrogen without using a lagoon, and 116 farmers did not test for nitrogen while using a lagoon. In total, $27 + 123 + 13 + 116 = 279$ farmers participated in the survey (there are no missing responses to any item).

Agresti and Liu (1999) provided the MRCV extension to testing for independence between single response categorical variables (SRCVs). This test, known as a test for simultaneous pairwise marginal independence (SPMI), involves determining whether each W_1, \dots, W_I is pairwise independent of each Y_1, \dots, Y_J . Our `MI.test()` function calculates their modified Pearson statistic as $X_S^2 = \sum_{i=1}^I \sum_{j=1}^J X_{S,i,j}^2$ where $X_{S,i,j}^2$ is the usual Pearson chi-square statistic used in this situation to test for independence in the 2×2 tables formed by each (W_i, Y_j) response combination. In our example, X_S^2 is the sum of 12

		Waste storage method			
		Lagoon	Pit	Natural Drainage	Holding tank
Contaminant	Nitrogen	27	16	2	2
	Phosphorous	22	12	1	1
	Salt	19	6	1	0

Table 1: Pairwise positive responses for the data in `farmer2`. While not recommended, this summary format is available through the `marginal.table()` function of **MRCV**.

pairwise marginal tests for independence. In general, X_S^2 does not have an asymptotic χ_{IJ}^2 distribution due to dependency among the $X_{S,ij}^2$. Rather, the asymptotic distribution is a linear combination of independent χ_1^2 random variables (Bilder and Loughin, 2004).

The `MI.test()` function offers three methods, available through its `type` argument, that can be used with X_S^2 or the $X_{S,ij}^2$ individual statistics to perform valid tests for SPMI. The `type = "boot"` argument value specifies the use of the nonparametric bootstrap to estimate the sampling distribution of X_S^2 under SPMI and to calculate an appropriate p-value using B resamples. In addition, two p-value combination methods—the product and minimum of p-values—are implemented to combine the p-values obtained from $X_{S,ij}^2$ and a χ_1^2 approximation. Details on these bootstrap approaches are given in Bilder and Loughin (2004). The `type = "rs2"` argument value applies a Rao-Scott second-order adjustment to X_S^2 and its sampling distribution. This procedure adjusts X_S^2 to match the first two moments of a chi-square random variable, asymptotically. Details on this approach are provided in Bilder and Loughin (2004) and Thomas and Decady (2004). Finally, the `type = "bon"` argument value simply applies a Bonferroni adjustment with each $X_{S,ij}^2$ and a χ_1^2 approximation. To implement all three methods, we can use the `type = "all"` argument value:

```
> set.seed(102211) # Set seed to replicate bootstrap results
> MI.test(data = farmer2, I = 3, J = 4, type = "all", B = 1999, plot.hist = TRUE)
```

Test for Simultaneous Pairwise Marginal Independence (SPMI)

Unadjusted Pearson Chi-Square Tests for Independence:

```
X^2_S = 64.03
X^2_S.ij =
      y1  y2  y3  y4
w1  4.93 2.93 14.29 0.01
w2  6.56 2.11 11.68 0.13
w3 13.98 0.00  7.08 0.32
```

Bootstrap Results:

```
Final results based on 1999 resamples
p.boot = 0.0005
p.combo.prod = 0.0005
p.combo.min = 0.001
```

Second-Order Rao-Scott Adjusted Results:

```
X^2_S.adj = 36.17
df.adj = 6.78
p.adj < 0.0001
```

Bonferroni Adjusted Results:

```
p.adj = 0.0019
p.ij.adj =
      y1  y2  y3  y4
w1 0.3163 1.0000 0.0019 1.0000
w2 0.1253 1.0000 0.0076 1.0000
w3 0.0022 1.0000 0.0934 1.0000
```

Figure 1 shows histograms from the bootstrap implementations. All of the methods provide strong evidence for rejecting SPMI. The $X_{S,ij}^2$ and corresponding Bonferroni adjusted p-values indicate a significant association for the (W_1, Y_3) , (W_2, Y_3) , and (W_3, Y_1) combinations.

Loglinear modeling

SPMI is only one possible association structure between MRCVs. Bilder and Loughin (2007) introduced a flexible loglinear modeling approach that allows researchers to consider alternative association structures somewhere between SPMI and complete dependence. Within this framework, a model under SPMI is given as

$$\log(\mu_{ab(ij)}) = \gamma_{ij} + \eta_{a(ij)}^W + \eta_{b(ij)}^Y \quad (1)$$

where $\mu_{ab(ij)}$ is the expected number of subjects who responded ($W_i = a, Y_j = b$) for $a, b \in \{0, 1\}$.

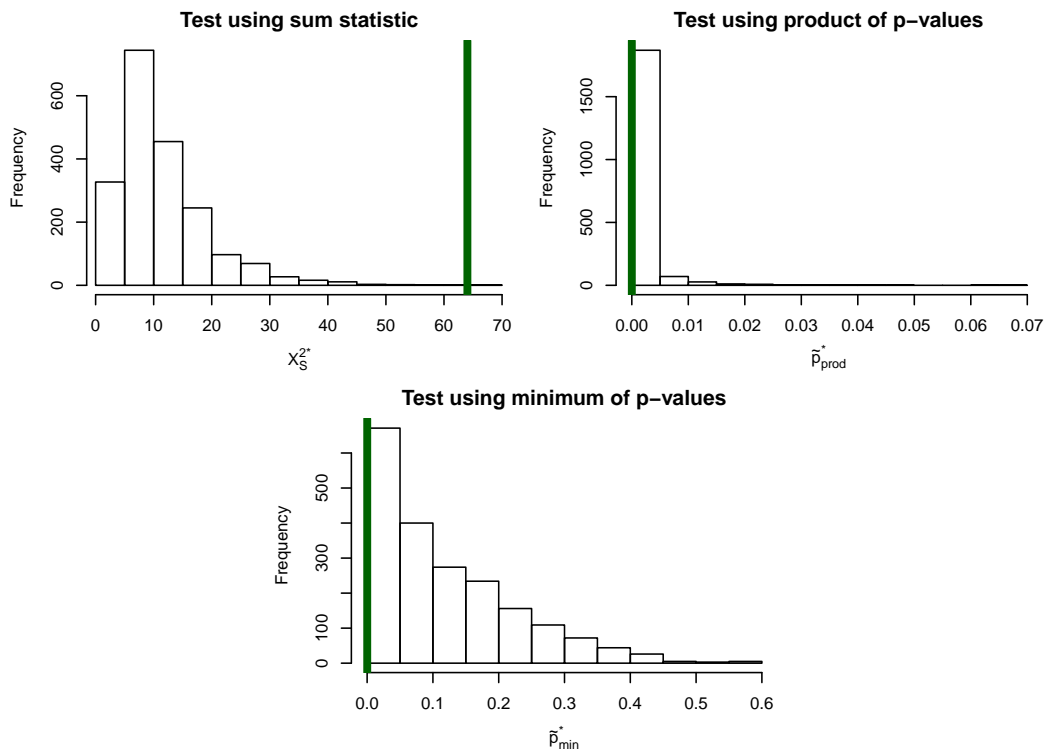


Figure 1: Histograms of the bootstrap estimated sampling distributions, where X_S^{2*} is the modified Pearson statistic calculated for a resample, and \tilde{p}_{prod}^* and \tilde{p}_{min}^* are the product and minimum p-value combination statistics, respectively, calculated for a resample. The vertical lines correspond to the statistics for the observed data.

The terms on the right side of the model are the same as for a loglinear model under independence between two SRCVs, where we have added a subscript (ij) to indicate a particular 2×2 table for (W_i , Y_j) within the pairwise item-response table. The usual constraints are placed on the model parameters to maintain identifiability.

Adding additional terms to Equation (1) leads to different types of association structures between the MRCVs. For example, adding λ_{ab} to Equation (1) produces a homogeneous association model (i.e., a model that implies equal odds ratios, not necessarily equal to 1, for each 2×2 table), adding $\lambda_{ab(i)}^W$ or $\lambda_{ab(j)}^Y$ to the homogeneous association model produces a W- or Y-main effects model, respectively, and adding both of these terms to the homogeneous association model produces a W- and Y-main effects model. The addition of a WY interaction term, $\lambda_{ab(ij)}^{WY}$, produces the saturated model.

The `genloglin()` function estimates the above models through a marginal estimation approach. Within `genloglin()`, a new data frame is created by converting the raw data into the pairwise item-response counts:

```
> item.response.table(data = farmer2, I = 3, J = 4, create.dataframe = TRUE)
  W  Y wi yj count
1 w1 y1 0 0  123
2 w1 y1 0 1  116
3 w1 y1 1 0   13
4 w1 y1 1 1   27
5 w1 y2 0 0  175

< output omitted >

48 w3 y4 1 1    0
```

The `glm()` function is subsequently called from within `genloglin()` to estimate a loglinear model to these counts. Rao-Scott adjustments are then applied to obtain valid large-sample standard error estimates. The model argument of `genloglin()` can take the names of "spmi", "homogeneous", "w.main", "y.main", "wy.main", and "saturated" to specify a particular model. Alternatively, a user-supplied formula allows for more flexibility by specifying the model in terms of W, Y, wi, yj, count, W1, ...

WI, and Y_1, \dots, Y_J , which we illustrate shortly. The `boot = TRUE` (the default) value for `genloglin()` specifies that resamples should be taken under the fitted model. We use the method of [Gange \(1995\)](#) for generating correlated binary data to perform semi-parametric bootstrap resampling in this case. These resamples are subsequently used for hypothesis tests, confidence intervals, and/or standardized residuals with our related method functions for objects returned by `genloglin()`.

We demonstrate the `genloglin()` function by estimating the Y-main effects model to the `farmer2` data, and then summarize the results using our `summary()` method function:

```
> set.seed(499077) # Set seed to replicate bootstrap results
> mod.fit <- genloglin(data = farmer2, I = 3, J = 4, model = "y.main", B = 1999,
+                      print.status = FALSE)
> summary(mod.fit)
```

Call:

```
glm(formula = count ~ -1 + W:Y + wi %in% W:Y + yj %in% W:Y + wi:yj + wi:yj %in% Y,
    family = poisson(link = log), data = model.data)
```

Deviance Residuals:

Min	1Q	Median	3Q	Max
-1.58007	-0.13272	0.00043	0.10282	0.79587

Coefficients:

	Estimate	RS SE	z value	Pr(> z)
Ww1:Yy1	4.83360	0.06535	73.969	< 2e-16 ***
Ww2:Yy1	4.85571	0.06387	76.023	< 2e-16 ***
Ww3:Yy1	4.87418	0.06314	77.199	< 2e-16 ***

< output omitted >

Null deviance: 25401.0663 Residual deviance: 5.8825
Number of Fisher Scoring iterations: 4

The `print.status` argument can be changed to `TRUE` (default) in order to print model fitting information while the function is running. Information typically provided by the `glm()` function can be extracted from `mod.fit`.

The formula argument within the `Call:` portion of the output displays an alternative way that the Y-main effects model could have been specified using variable names. For a model under SPMI (Equation (1)), the syntax `-1 + W:Y + wi %in% W:Y + yj %in% W:Y` specifies an ordinary loglinear model under independence *within* each 2×2 table formed by the (W_i, Y_j) pairs; i.e., the intercept ($W:Y$), "row effect" ($wi \%in\% W:Y$), and "column effect" ($yj \%in\% W:Y$) terms. Note that the addition of `wi:yj %in% W:Y` would then lead to a saturated loglinear model within the 2×2 tables. Instead, the addition of `wi:yj + wi:yj %in% Y` allows for the associations to vary across the items in Y (waste storage) but to be the same across items in W (contaminant).

The deviance values in the output should not be used with chi-square distributional approximations to construct traditional model comparison tests. Instead, our `anova()` method function offers bootstrap and Rao-Scott second-order adjustments (`type = "boot"` and `type = "rs2"`, respectively, or `type = "all"` for both methods) to obtain appropriate tests for comparing the model specified in `genloglin()` to an alternative model given by its `model.HA` argument. Comparing the Y-main effects model to the saturated model shows moderate evidence of lack-of-fit:

```
> anova(object = mod.fit, model.HA = "saturated", type = "all")
```

Model comparison statistics for
H0 = y.main
HA = saturated

Pearson chi-square statistic = 5.34
LRT statistic = 5.88

Second-Order Rao-Scott Adjusted Results:
Rao-Scott Pearson chi-square statistic = 10.85, df = 5.23, p = 0.0624
Rao-Scott LRT statistic = 11.96, df = 5.23, p = 0.0409

Bootstrap Results:

Final results based on 1999 resamples
 Pearson chi-square p-value = 0.0385
 LRT p-value = 0.0255

Our `residuals()` method function provides Pearson standardized residuals, where bootstrap or asymptotic standard errors can be used in their formation. For the Y-main effects model, we find that lack-of-fit occurs for the (W_3, Y_1) association. This suggests the need to estimate a new model that explicitly accounts for the heterogeneity:

```
mod.fit.w3y1 <- genloglin(data = farmer2, I = 3, J = 4, model = count ~ -1 + W:Y +
  wi %in% W:Y + yj %in% W:Y + wi:yj + wi:yj %in% Y +
  wi:yj %in% W3:Y1, B = 1999)
```

where the `wi:yj %in% W3:Y1` term forces a perfect fit to the (W_3, Y_1) association while still maintaining a Y-main effects model elsewhere.

Once an appropriate model has been identified, our `predict()` method function can be used to obtain observed and model-estimated odds ratios with corresponding asymptotic and bootstrap BC_a (Efron, 1987) confidence intervals. These odds ratios help to facilitate interpretation of the association among items between the two MRCVs.

Summary

The equivalents of many traditional categorical data analysis methods are implemented within our package in the context of MRCVs. We demonstrated a few of the package's primary functions for analyzing the association between two MRCVs. While not shown here, these functions can be used to analyze MRCVs in other settings. For instance, tests for *multiple marginal independence* (MMI; Agresti and Liu, 1999) between an MRCV and an SRCV can be performed by `MI.test()`, where the `I` argument is set to a value of 1. An example is given within the help file for this function. Additionally, the MRCV package can be used to analyze the association between three MRCVs. For example, Bilder and Loughin (2007) discuss a third "choose all that apply" question asked of the swine farmers that relates to the farmers' sources of veterinary information. We show in the help file for `genloglin()` how to estimate a generalized loglinear model for this setting.

Agresti and Liu (1999, 2001) show how to take advantage of many commonly used modeling methods (e.g., generalized linear mixed models) for MRCV data. Most of these methods have disadvantages to their use—for example, standard generalized linear mixed models induce a positive correlation between binary responses within subjects, but a negative correlation can occur with MRCV data. Their recommended modeling method, a generalized loglinear model fit through generalized estimating equation (GEE) methodology, can work reasonably well in very large sample sizes (Bilder et al., 2000). The help file for `MI.test()` shows how to use functions in the `geepack` package (Yan et al., 2012) to estimate this model and then subsequently test for MMI via a Wald test.

We envision future additions to the package that will allow for extensions to other situations. For example, "choose all that apply" questions are often asked in complex survey sampling settings. Bilder and Loughin (2009) propose using the same generalized loglinear models, where now different Rao-Scott adjustments are needed to take into account the sampling design. Also, MRCV data can arise over a longitudinal setting, and Suesse and Liu (2013) propose the use of GEE methodology to fit models for this situation. Finally, Nandram et al. (2009) offer a Bayesian perspective to the MMI testing problem. Due to the ubiquitous nature of "choose all that apply" type data formats, we expect there to be many other unique settings where new statistical methods need to be developed. We encourage readers to contact us about their novel methods and/or interest in collaboration.

Bibliography

- A. Agresti and I. Liu. Modeling a categorical variable allowing arbitrarily many category choices. *Biometrics*, 55(3):936–943, 1999. [p144, 145, 149]
- A. Agresti and I. Liu. Strategies for modeling a categorical variable allowing multiple category choices. *Sociological Methods & Research*, 29(4):403–434, 2001. [p149]
- C. Bilder and T. Loughin. On the first-order Rao-Scott correction of the Umesh-Loughin-Scherer statistic. *Biometrics*, 57(4):1253–1255, 2001. [p145]
- C. Bilder and T. Loughin. Testing for marginal independence between two categorical variables with multiple responses. *Biometrics*, 60(1):241–248, 2004. [p144, 146]

- C. Bilder and T. Loughin. Modeling association between two or more categorical variables that allow for multiple category choices. *Communications in Statistics—Theory and Methods*, 36(2):433–451, 2007. [p144, 146, 149]
- C. Bilder and T. Loughin. Modeling multiple-response categorical data from complex surveys. *The Canadian Journal of Statistics*, 37(4):553–570, 2009. [p149]
- C. Bilder, T. Loughin, and D. Nettleton. Multiple marginal independence testing for pick any/c variables. *Communications in Statistics—Simulation and Computation*, 29(4):1285–1316, 2000. [p149]
- C. Coombs. *A Theory of Data*. John Wiley & Sons, New York, 1964. [p144]
- B. Efron. Better bootstrap confidence intervals. *Journal of the American Statistical Association*, 82(397):171–185, 1987. [p149]
- S. Gange. Generating multivariate categorical variates using the iterative proportional fitting algorithm. *The American Statistician*, 49(2):134–138, 1995. [p148]
- H. Kantarjian, J. Cortes, F. Guilhot, A. Hochhaus, M. Baccarani, and L. Lokey. Diagnosis and management of chronic myeloid leukemia: A survey of American and European practice patterns. *Cancer*, 109(7):1365–1375, 2007. [p144]
- N. Koziol and C. Bilder. *MRCV: Methods for Analyzing Multiple Response Categorical Variables*, 2014. URL <http://CRAN.R-project.org/package=MRCV>. R package version 0.3-1. [p144]
- P. Lemons, J. Sedinger, M. Herzog, P. Gipson, and R. Gilliland. Landscape effects on diets of two canids in northwestern Texas: A multinomial modeling approach. *Journal of Mammalogy*, 91(1):66–78, 2010. [p144]
- T. Loughin and P. Scherer. Testing for association in contingency tables with multiple column responses. *Biometrics*, 54(2):630–637, 1998. [p144]
- B. Nandram, M. Toto, and M. Katzoff. Bayesian inference for a stratified categorical variable allowing all possible category choices. *Journal of Statistical Computation and Simulation*, 79(2):161–179, 2009. [p149]
- Office of Management and Budget. Revisions to the standards for the classification of federal data on race and ethnicity. *Federal Register*, 62:58781–58790, 1997. [p144]
- B. Riegel, D. Moser, M. Powell, T. Rector, and E. Havranek. Nonpharmacologic care by heart failure experts. *Journal of Cardiac Failure*, 12(2):149–153, 2006. [p144]
- S. Riemer, B. Wright, and R. Brown. Food habits of Steller sea lions (*Eumetopias jubatus*) off Oregon and northern California, 1986–2007. *Fishery Bulletin*, 109(4):369–381, 2011. [p144]
- T. Suesse and I. Liu. Modelling strategies for repeated multiple response data. *International Statistical Review*, 81(2):230–248, 2013. [p149]
- D. Thomas and Y. Decady. Testing for association using multiple response survey data: Approximate procedures based on the Rao-Scott approach. *International Journal of Testing*, 4(1):43–59, 2004. [p146]
- B. Wright. Use of chi-square tests to analyze scat-derived diet composition data. *Marine Mammal Science*, 26(2):395–401, 2010. [p144]
- J. Yan, S. Hojsgaard, and U. Halekoh. *geepack: Generalized Estimating Equation Package*, 2012. URL <http://CRAN.R-project.org/package=geepack>. R package version 1.1-6. [p149]

Natalie A. Koziol
 Department of Statistics; Department of Educational Psychology
 University of Nebraska-Lincoln
 Lincoln, NE, United States
nak371@gmail.com

Christopher R. Bilder
 Department of Statistics
 University of Nebraska-Lincoln
 Lincoln, NE, United States
chris@chrisbilder.com

Archiving Reproducible Research with R and Dataverse

by Thomas J. Leeper

Abstract Reproducible research and data archiving are increasingly important issues in research involving statistical analyses of quantitative data. This article introduces the `dv` package, which allows R users to publicly archive datasets, analysis files, codebooks, and associated metadata in Dataverse Network online repositories, an open-source data archiving project sponsored by Harvard University. In this article I review the importance of data archiving in the context of reproducible research, introduce the Dataverse Network, explain the implementation of the `dv` package, and provide example code for archiving and releasing data using the package.

Data archiving and reproducible research

Reproducible research involves the careful, annotated preservation of data, analysis code, and associated files, such that statistical procedures, output, and published results can be directly and fully replicated (see, for example, King, 1995; Boyer, 2003). R provides an increasingly large set of tools for engaging in reproducible research practices. Perhaps most prominent is `knitr` (Xie, 2013), which expands upon Sweave (Leisch, 2002) to help R users produce fully reproducible research reports for a variety of markup languages. Gandrud (2013b) and others have advocated for the widespread use of reproducible research tools, but a major shortcoming of the existing set of tools is a means of easily archiving reproducible research files for use by others. While it is easy to privately archive files for reproducing one's own work later on, making study files publicly available is not something readily supported by R or other existing tools.

A reproducible research workflow is most valuable when it allows others to replicate results and reproduce published results precisely. Thus complete records of all data and analyses need to be publicly and persistently available via a stable, freely available archive. Gandrud (2013a,b) has advocated for the use of Dropbox and GitHub for fulfilling this archiving role. Both sites are supported, to varying extents, by existing R packages: `rDrop` (Ram, 2012) for Dropbox and `rgithub` (Scheidtger, 2013) for GitHub. However, Dropbox is not designed for persistent archiving, depends on assumptions about the future availability of a proprietary, closed-source web service, and offers no platform for finding archived data (thus introducing a further dependence on the publication of a Dropbox link somewhere else on the web). GitHub, which integrates with the git version-control system (which many R users may already invoke as part of their existing workflow), is helpful for supporting an ongoing (and possibly collaborative) reproducible research workflow but is not designed to be a persistent data archive.

Thus researchers remain in need of a service dedicated to the persistent preservation of data with sophisticated metadata support to allow others to easily find, understand, and use archived files. One such option is The Dataverse Network, a free-to-use, open-source data archive sponsored and developed by the Institute for Quantitative Social Science at Harvard University (Altman et al., 2001; King, 2007).¹ The next section introduces The Dataverse Network and the remainder of the article describes how to utilize the archive directly from R via the `dv` package.

The Dataverse Network

The Dataverse Network is a server application that hosts collections of studies, called dataverses.² As an open-source application, The Dataverse Network has many instances (i.e., hosted by different institutions) and each implementation, e.g., the Harvard IQSS Dataverse Network, hosts many dataverses controlled by individuals, institutions, and academic journals. Though there are few The Dataverse Network instances (about 10 worldwide), each of these — especially those hosted by Harvard University and the Odum Institute at the University of North Carolina–Chapel Hill — hosts thousands of individual dataverses containing a collective hundreds of thousands of study listings. Consequently, The Dataverse Network is an increasingly prominent repository of data and associated files and is the default archive for many journals that currently require public archiving of replication files as a condition of publication. To use The Dataverse Network, one needs to create an account. To

¹ Another alternative is <http://figshare.com/>, which is already supported by `rfigshare`.

² Hereafter, the small-d “dataverse” refers to a collection of studies stored with The Dataverse Network and “The Dataverse Network” refers to the general web application.

release studies, one also needs to create a personal dataverse collection associated with the account. Accounts are tied to specific The Dataverse Network instances, so in this article we will focus for simplicity's sake on the [The Harvard Dataverse Network](#).

Each study archived in a dataverse contains, at a minimum, an identifying title, but may additionally include an array of metadata and files. This means The Dataverse Network can be used to store not just a free-standing data file, but associated files in almost any format and one might include files such as data, codebooks, analysis replication files, statistical packages, questionnaires, experimental materials, and so forth. Data uploaded to a dataverse is converted to a generic tabular format and can be subsequently downloaded into one of several common formats (e.g., Stata, S-Plus, R, and tab-delimited).

Once created, a study is given a global identifier in the form of either a Handle,³ or DataCite DOI,⁴ digital object identifiers that uniquely and globally identify the study. Studies are version controlled when new files are added or when metadata is modified, and previous versions remain persistently accessible. Furthermore, when data files are modified, the study identifier is updated with additional version information in the form of Universal Numeric Fingerprint (UNF) ([Altman and King, 2007](#); [Altman, 2008](#)). The UNF is a format-independent hash of a data file, which provides a unique identifier for a dataset without revealing the content of that dataset, such that any change to the underlying data yields a new UNF. This means that data stored in a dataverse can be cited (e.g., in scholarly publications) not only by their handle but also by their specific version using the UNF.⁵ Reproducible research efforts can thus be enhanced by use not only of the same named dataset but the exact same version of that dataset, a major advantage over storing data in other types of archive.⁶ Via a UNF, data users can thus check a dataset they possess against one cited in a research publication or stored in a dataverse.⁷

Archiving data and files using The Dataverse Network also means that those files can be made available to others. The Dataverse Network settings offer fine-grained control over access to archived studies, meaning a study collection can be regulated anywhere from completely publicly accessible, to accessible only to registered users, to accessible only to whitelisted users. Public data then becomes easily searchable by others based on any of the metadata associated with the study files, a major advantage over archiving on any other website.⁸ The supported metadata elements include almost all imaginable aspects of a data collection, include those provided by the Dublin Core standard,⁹ such as creator, subject, description, date, type, rights,¹⁰ citation, etc. Additional metadata elements are also supported and Dataverse additionally supplies metadata in the Data Documentation Initiative (DDI) format,¹¹ a comprehensive metadata schema for quantitative data, especially for the social sciences.

The dvn package

The **dvn** package provides a lightweight wrapper for two Application Programming Interfaces (APIs) provided by The Dataverse Network. The first API — called the Data Sharing API — is essentially a search utility for locating existing studies within a Dataverse Network instance using simple HTTP GET requests. The second API — called the Data Deposit API — is a bit more complicated. It is built on the SWORD (Simple Web-service Offering Repository Deposit) protocol,¹² an open-source standard for the deposit of digital records.¹³ This API relies on HTTP requests to GET, PUT, POST, and DELETE resources on a Dataverse server. Access to both APIs is implemented through **RCurl** ([Temple Lang, 2012a](#)) and responses are parsed with **XML** ([Temple Lang, 2012b](#)). From the user perspective, **dvn** masks the distinction between the two APIs, offering functions both for conducting data searches and for archiving data, code, and associated metadata.

³<http://hdl.net/>

⁴<http://www.datacite.org/>

⁵The version-controlled Handle thus serves as a complete data citation, referencing a specific version of a specific data file (see <http://thedata.org/citation/standard>). A description of (current) Version 5 of the UNF algorithm is available at <http://thedata.org/book/unf-version-5-0>.

⁶Of course, a version-controlled repository (e.g., on GitHub), would also create a hash for objects.

⁷Functions necessary to calculate a UNF signature on a dataframe are provided by **UNF** ([Leeper, 2013](#)).

⁸The need for metadata in the reproducible research workflow has previously been identified by [Gentleman and Temple Lang \(2004\)](#).

⁹<http://dublincore.org/documents/dcmi-terms/>

¹⁰The Dataverse Network supports any user-defined licensing terms for data use. For a discussion of issues related to data licensing, see [Stodden \(2009\)](#).

¹¹<http://www.ddialliance.org/Specification/>

¹²<http://swordapp.org/>

¹³As an example, the SWORD standard is also used by Open Journal Systems for managing articles through the review and publication process and for making them publicly available.

Thus **dvn** adds to an increasingly large number of packages that connect R to web-based services,¹⁴ such as those built by the ROpenSci project.¹⁵ Perhaps the most proximate existing R package to **dvn** is **rfigshare** (Boettiger et al., 2013), which allows R users to persistently store files on <http://figshare.com/>. **rdryad** (Chamberlain et al., 2013) and **OAIHarvester** (Hornik, 2013) — which can be used to harvest metadata records from <http://datadryad.org/> or, in the latter case, any public repository that supports the OAI-PMH standard¹⁶ — share commonalities with The Dataverse Network Data Sharing API but neither allows users to upload their own files.

dvn will remain under active development for the foreseeable future and will track any modifications made to The Dataverse Network APIs. Bug reports, code pull requests, and other suggestions are welcome on the **dvn** homepage at <https://github.com/rOpenSci/dvn>. The next two sections describe functionality of the package in its current form.

Data and metadata search with dvn

The search functionality of **dvn** can be used to easily locate archived data in any public Dataverse Network. **dvn** defaults to providing access to The Harvard Dataverse Network (located at <https://thedata.harvard.edu/dvn/>), but this can be changed in each function call or globally using options:

```
# use Odum Institute Dataverse
options(dvn = 'https://arc.irss.unc.edu/dvn/')
# use Harvard IQSS Dataverse
options(dvn = 'https://thedata.harvard.edu/dvn/')
```

for any valid Dataverse Network.¹⁷ With a Dataverse Network selected, we can easily search that network for studies using any metadata field. `dvSearchField` provides a list of available metadata search fields. These can be used to search for studies by various authors or on various topics using `dvSearch` (which accepts a boolean logic):

```
dvSearch(list(authorName = "leeper"))
dvSearch(list(title = "Denmark", title = "Sweden"), boolean = "OR")
dvSearch("Puppies")
```

Calling `dvSearch` with a single character argument searches all metadata fields simultaneously using a boolean OR logic. The result is (presently) a one-column dataframe listing `objectId` values, which represent the global identifier for each study.¹⁸ Here is an example using our final search from above:

```
> dvSearch('puppies')
2 search results returned

      objectId
1 hdl:1902.1/21521
2 hdl:1902.1/21522
```

An `objectId` can then be used to retrieve detailed metadata available for a study. Metadata is available in two formats: Dublic Core (DC) and Data Documentation Initiative (DDI).¹⁹ DC is an almost universally adopted, minimalist metadata schema widely used in information science. DDI is a more sophisticated format designed initially for social science data applications that can include considerably more metadata information.²⁰ By default, Dataverse and **dvn** return the more comprehensive DDI format. To request DC, one needs to specify it using the `format.type` argument:

```
dvMetadata("hdl:1902.1/21964") # return DDI (by default)
dvMetadata("hdl:1902.1/21964", format.type="oai_dc") # return DC
```

Currently `dvMetadata` returns metadata as a character string with S3 class `"dvMetadata"` because both DC and DDI output can be quite extensive and are better viewed in a text editor than in the

¹⁴<http://cran.r-project.org/web/views/WebTechnologies.html>

¹⁵<http://ropensci.org/>

¹⁶<http://www.openarchives.org/pmh/>

¹⁷See <http://thedata.org/book/dataverse-networks-around-world> for a complete listing of public Dataverse Networks. Harvard also hosts a demo server for experimenting with The Dataverse Network: <http://dvn-demo.iq.harvard.edu/dvn/>.

¹⁸Dataverse has used both Handles and DOIs as global identifiers, thus the generic and forward-compatible `objectId` terminology.

¹⁹For a given study, it is possible to check the availability of different metadata formats using `dvMetadataFormats`.

²⁰A crosswalk mapping metadata elements from DC, DDI, and the Dataverse web interface is available at <http://thedata.harvard.edu/guides/dataverse-api-main.html#data-deposit-api>.

R console. For similar reasons, **dvn** does not fully parse the DDI or DC XML. Instead, the wrapper function `dvExtractFileIds` extracts relevant information from the DDI formatted metadata.²¹ Here's an example:

```
files <- dvExtractFileIds(dvMetadata("hdl:1902.1/21964"))
> files[, c('fileName', 'fileId')]
      fileName  fileId
1  study2-replication-analysis.r 2341713
2  study1-replication-analysis.r 2341709
3          coeftpaste.r 2341888
4      expResults.r 2341889
5      Study 2 Codebook.docx 2341712
6 study2-data-final-2012-06-08.csv 2341711
7 study1-data-final-2012-06-08.csv 2341710
8      Study 2 Webpages.zip 2341714
9      Study 1 Webpages.zip 2341715
```

With a `fileId` returned by `dvExtractFileIds`, one can see the formats available for a file (e.g., data can be downloaded in a number of formats), using `dvDownloadInfo`:

```
> dvDownloadInfo(files$fileId[1])
File Name:      study2-replication-analysis.r
File ID:        2341713
File Type:      text/plain; charset=US-ASCII
File Size:      15699
Authentication: anonymous
Direct Access?  false
```

Terms of Use apply.

Access Services:

	serviceName	serviceArgs	contentType	serviceDesc
1	termsofuse	TermsOfUse=true	text/plain	
2	bundleTOU	package=WithTermsOfUse	application/zip	
1	Terms of Use/Access Restrictions associated with the data file			
2	Data File and the Terms of Use in a Zip archive			

If allowed by Terms of Use, `dvDownload` can be used to download a file directly into memory. Otherwise the output of `dvDownload` will advise the user to download the data through a browser using the URI returned by `dvExtractFileIds`:

```
> dvDownload(files$fileId[1])
Error in dvDownload(files$fileId[1]) :
  Terms of Use apply.
```

Data cannot be accessed directly...

try browsing URI from `dvExtractFileIds(dvMetadata())`

```
> files$URI[1]
[1] http://thedata.harvard.edu/dvn/dv/leeper/FileDownload/study2-
replication-analysis.r?fileId=2341713
```

Depositing data with dvn

While searching for extant data files is a helpful feature to have within R, the **dvn** wrappers for the Data Deposit API offer much more useful tools for archiving a reproducible research project. The basic workflow involves functions to:

1. create a study using appropriate metadata (`dvCreateStudy`)
2. add one or more files to that study (`dvAddFile`)
3. release the study to the public (`dvReleaseStudy`)

²¹Another wrapper extracts Terms of Use from DDI metadata and opens those terms of use as a temporary HTML file: `dvTermsOfUse(dvMetadata(objectid))`.

Use of the Data Deposit functions requires authentication, which is currently provided by HTTP basic authentication, relying on the username and password associated with a Dataverse Network user account. The username and password can be included atomically in each function call or stored globally (to conserve typing and allow the sharing of code without credentials embedded in each function call):

```
options(dvn.user = "username")
options(dvn.pwd = "password")
```

As a simple check of authentication, one can call `dvServiceDoc()` with no arguments, which (if authentication is successful) returns a brief statement confirming the Dataverse implementation and user account in use along with a list of available dataverse collections, or otherwise reports an HTTP 403 (Forbidden) code.

To create a study, we simply need to call `dvCreateStudy` with some associated metadata in Dublin Core format. We can either build a metadata file manually or use `dvBuildMetadata` to build that XML for us. The only required field is "title." The built metadata can then be supplied to `dvCreateStudy`, additionally specifying a named dataverse to which the user has access:²²

```
m <- dvBuildMetadata(title = "My Study")
s <- dvCreateStudy("mydataverse", m)
```

If successful, the response is a simple summary of the created study. Included in the response is the new study's `objectId`, which uniquely identifies the study. We can use either the `objectId` or the return value of `dvCreateStudy` (an object of class "dvStudyAtom") to add files and release the study.

We can add files in a number of ways and in a number of formats. For example, we can add one or more files simply by specifying their filenames in `dvAddFile`. The filename is used to list the file in the study. We can also send multiple files in a compressed zip directory, which The Dataverse Network application will automatically unpack.²³ Another option is to directly send a named dataframe directly from memory:

```
# add files and release study using `objectId`
dvAddFile(s, "mydata.zip")
# or add multiple files:
dvAddFile(s, c("file1.csv", "file2.txt"))
# or add R dataframes as files:
mydf <- data.frame(x = 1:10, y = 11:20)
dvAddFile(s, dataframe = "mydf")
```

Adding files is non-destructive, so we can add files all at once in a single .zip directory or we can add them sequentially. The Dataverse Network does not overwrite an existing file. Attempting to do so will result in an error and `dvAddFile` will return NULL. If a file fails to upload successfully, `dvAddFile` will return an error, and if a data file uploads successfully but then fails to decompress or is in an unrecognizable format, you will receive an email to the account affiliated with your dataverse informing you of the failure.

With metadata and files added, we can release the study using `dvReleaseStudy`. Before doing so, it may be reasonable to check the study using `dvStudyStatement` to see a short summary of study contents:

```
> dvStudyStatement(s)
Study author: Unknown
Study title: My Study
ObjectId: doi:10.7910/DVN/ILAJ0
Study URI: https://thedata.harvard.edu/dvn/api/data-deposit/v1/swordv2/edit/study/doi:10.7910/DVN/ILAJ0
Last updated: 2013-12-02T09:56:07.298Z
Status: DRAFT
Locked? false
Files:
  src
1 https://thedata.harvard.edu/dvn/api/data-deposit/v1/swordv2/edit-media/file/2349641/mydf.tab
  type                updated                fileId
1 text/tab-separated-values 2013-12-02T09:56:16.227Z 2349641
```

²²Access can be checked by examining the output of `dvServiceDoc`.

²³An optional category argument additionally allows you to group files into categories, such as "data" and "code."

The above example output shows a variety of information, including the lack of a study author, the state of the study as DRAFT (meaning the current version is unreleased), and a list of the included files (we see the mydf object uploaded above). If satisfied, we can release the study:

```
dvReleaseStudy(s)
```

When a study is released, it becomes immediately accessible via its persistent Handle or DOI. If changes are made to the dataverse (either metadata or files), these remain in draft form until the dataverse is released again. Each release is persistently available unless the entire study collection is deleted.²⁴

Before releasing a study, we can delete it from The Dataverse Network server using `dvDeleteStudy`. Once a study is released, it cannot be deleted, but access to it can be revoked. Thus, calling `dvDeleteStudy` only deletes studies that are unreleased. Calling `dvDeleteStudy` on a released study revokes public access, but a persistent page will remain at the study URL noting that a study was previously released. Similarly, it is possible to delete files using `dvDeleteFile` with an argument specifying the `fileId` (which we can extract from `dvStudyStatement`):

```
> d <- dvStudyStatement(s)
> d$files$fileId
[1] "2349642" "2349641"
> dvDeleteFile("2349642")
Operation appears to have succeeded.
[1] ""
```

If a file deletion is successful, **dvN** will report a success message and `dvDeleteFile` will return an empty character string.²⁵

We can also modify a study's metadata using `dvEditStudy`, supplied with a metadata listing (e.g., as returned by `dvBuildMetadata`).²⁶

```
> m2 <- dvBuildMetadata(title = 'An Actual Title', creator = 'Me', date = '2013')
> dvEditStudy(s, m2)
Citation:      Me, 2013, "An Actual Title", http://dx.doi.org/10.7910/DVN/ILAJ0 V1
ObjectId:      doi:10.7910/DVN/ILAJ0
Study URI:      https://thedata.harvard.edu/dvn/api/data-deposit/v1/swordv2/edit/study/
                  doi:10.7910/DVN/ILAJ0
Generated by:   http://www.swordapp.org/ 2.0
```

The response reflects the updated metadata. If changes are made to a study (files are added or deleted, or metadata is modified), those changes will be visible via **dvN** as a DRAFT study version, but will not be public. To make them public, simply release the study again with `dvReleaseStudy`. Both study versions will remain accessible via The Dataverse Network web browser interface.

Once we've created one or more studies (regardless of whether they are released), we can view them using `dvUserStudies` for a named dataverse or from a `dvServiceDoc` response:

```
dvUserStudies("mydataverse")
# or:
dvUserStudies(dvServiceDoc())
```

Thus while we only need three core functions (`dvCreateStudy`, `dvAddFile`, and `dvReleaseStudy`) to complete a reproducible research workflow, the other functions supplied by **dvN** allow a great degree of control over the appearance and contents of a study.

Conclusion

As Gandrud (2013b) has demonstrated, R is a powerful tool in the reproducible research workflow. Yet a missing element of that workflow to-date has been the easy ability to store research files in a persistent, public data repository designed for reproducible research. The Dataverse Network is a free, open-source service explicitly dedicated to permanently storing data, with implementations hosted by

²⁴Note, however, that The Dataverse Network APIs do not currently offer access to previous study versions, though they are accessible via the web interface.

²⁵Note, however, that attempting to delete a file that does not exist returns no error message from the SWORD server.

²⁶Note, however, that this function completely overwrites all metadata in a study, effectively deleting the current metadata and repopulating it with the supplied metadata.

a variety of institutions (mostly research universities) offering additional choice over whom to trust with the persistent storage of study records. By automatically providing study records a unique URL, data stored using The Dataverse Network has a version-controlled and persistent identifier that can be cited by subsequent users of the data, continuing the reproducible workflow process beyond the phase of data production and initial analysis. With the popularity of The Dataverse Network increasing, it is a logical choice for archiving one's reproducible data and analysis files. **dvn** thus provides R users with the tools necessary to quickly and easily integrate data archiving into the reproducible research workflow.

Bibliography

- M. Altman. A fingerprint method for scientific data verification. In T. Sobh, editor, *Advances in Computer and Information Sciences and Engineering*, chapter 57, pages 311–316. Springer Netherlands, Netherlands, 2008. ISBN 978-1-4020-8740-0. doi: 10.1007/978-1-4020-8741-7_57. URL http://link.springer.com/chapter/10.1007/978-1-4020-8741-7_57. [p152]
- M. Altman and G. King. A proposed standard for the scholarly citation of quantitative data. *D-Lib Magazine*, 13(3/4):1, 2007. doi: 10.1045/march2007-altman. URL <http://www.dlib.org/dlib/march07/altman/03altman.html>. [p152]
- M. Altman, L. Andreev, M. Diggory, G. King, A. Sone, S. Verba, D. L. Kiskis, and M. Krot. A digital library for the dissemination and replication of quantitative social science research: The virtual data center. *Social Science Computer Review*, 19(4):458–470, 2001. [p151]
- C. Boettiger, S. Chamberlain, K. Ram, and E. Hart. *rfigshare: an R interface to figshare.com*, 2013. URL <http://cran.fhcrc.org/web/packages/rfigshare/index.html>. R package version 0.2-6. [p153]
- M. A. Boyer. Symposium on replication in international studies research. *International Studies Perspectives*, 4:72–107, 2003. [p151]
- S. Chamberlain, C. Boettiger, and K. Ram. *rdryad: Interface to the API for Dryad*, 2013. URL <http://cran.r-project.org/web/packages/rdryad/index.html>. R package version 0.1.1. [p153]
- C. Gandrud. Github: A tool for social data development and verification in the cloud. *The Political Methodologist*, 20(2):7–16, 2013a. URL http://polmeth.wustl.edu/methodologist/tpm_v20_n2.pdf. [p151]
- C. Gandrud. *Reproducible Research with R and RStudio*. Chapman and Hall/CRC, Boca Raton, FL, 2013b. ISBN 9781466572843. [p151, 156]
- R. Gentleman and D. Temple Lang. Bioconductor project statistical analyses and reproducible statistical analyses and reproducible. Unpublished paper, 2004. [p152]
- K. Hornik. *OAIHarvester: Harvest Metadata Using OAI-PMH v2.0*, 2013. URL <http://cran.r-project.org/web/packages/OAIHarvester/index.html>. R package version 0.1-6. [p153]
- G. King. Replication, replication. *PS: Political Science & Politics*, 28(3):444–451, Sept. 1995. [p151]
- G. King. An introduction to the Dataverse Network as an infrastructure for data sharing. *Sociological Methods & Research*, 36(2):173–199, Nov. 2007. ISSN 0049-1241. doi: 10.1177/0049124107306660. URL <http://smr.sagepub.com/cgi/doi/10.1177/0049124107306660>. [p151]
- T. J. Leeper. *UNF: Tools for creating universal numeric fingerprints for data*. Aarhus University, Aarhus, Denmark, 2013. URL <https://github.com/leeper/UNF>. R package version 0.1. [p152]
- F. Leisch. Sweave: Dynamic generation of statistical reports using literate data analysis. *Compstat 2002 - Proceedings in Computational Statistics*, (69):575–580, 2002. URL <http://www.stat.uni-muenchen.de/~leisch/Sweave/>. [p151]
- K. Ram. *rDrop: Programmatic interface to Dropbox*, 2012. URL <https://github.com/karthik/rDrop>. R package version 0.3-0. [p151]
- C. Scheidegger. *rgithub: R bindings for the Github API*, 2013. URL <https://github.com/cscheid/rgithub>. R package version 0.9.5. [p151]
- V. Stodden. Enabling reproducible research: Licensing for scientific innovation. *International Journal of Communications Law & Policy*, 13:1–25, 2009. [p152]

- D. Temple Lang. *RCurl: General network (HTTP/FTP/...) client interface for R*, 2012a. URL <http://cran.r-project.org/web/packages/RCurl/index.html>. [p152]
- D. Temple Lang. *XML: Tools for parsing and generating XML within R and S-Plus*, 2012b. URL <http://cran.r-project.org/package=XML>. [p152]
- Y. Xie. *knitr: A general-purpose package for dynamic report generation in r*, 2013. URL <http://cran.r-project.org/web/packages/knitr/index.html>. [p151]

Thomas J. Leeper
Department of Political Science and Government
Aarhus University
Denmark
thosjleeper@gmail.com

oligoMask: A Framework for Assessing and Removing the Effect of Genetic Variants on Microarray Probes

by Daniel Bottomly, Beth Wilmot and Shannon K. McWeeney

Abstract As expression microarrays are typically designed relative to a reference genome, any individual genetic variant that overlaps a probe's genomic position can possibly cause a reduction in hybridization due to the probe no longer being a perfect match to a given sample's mRNA at that locus. If the samples or groups used in a microarray study differ in terms of genetic variants, the results of the microarray experiment can be negatively impacted. The **oligoMask** package is an R/SQLite framework which can utilize publicly available genetic variants and works in conjunction with the **oligo** package to read in the expression data and remove microarray probes which are likely to impact a given microarray experiment prior to analysis. Tools are provided for creating an SQLite database containing the probe and variant annotations and for performing the commonly used RMA preprocessing procedure for Affymetrix microarrays. The **oligoMask** package is freely available at <https://github.com/dbottomly/oligoMask>.

Introduction

It has been observed that for mRNA microarrays from a given sample, genetic differences of that sample relative to the probe sequences can affect hybridization to short oligonucleotide probes resulting in false positives or negatives depending on the experimental and array design (Walter et al., 2007; Alberts et al., 2007). Several approaches currently exist to identify and flag/remove probes that have hybridization artifacts due to genetic variants. Removal of probes based on pre-defined genetic variant databases is one such approach (Benovoy et al., 2008). Software (Kumari et al., 2007) and databases (Duan et al., 2008) allowing the interrogation of the relationships between microarray probes and single nucleotide variants have been described. The R package **CustomCDF** (Dai et al., 2005) is an example of this approach that removes probes from an environment formed from the Affymetrix chip description file (CDF) prior to analysis. One potential limitation of the CDF filtering approach is that for some of the more recent arrays platforms such as the Affymetrix Gene or Exon arrays the use of such environments has been superseded (e.g. the SQLite databases in the **oligo** (Carvalho and Irizarry, 2010) package or ROOT scheme files as in the **xps** (Stratowa et al., 2013) package).

In addition, the actual expression data itself can be interrogated to identify and mask out variants. R packages exist to effectively deal with a two group comparison between several strains or species through procedures based mainly on the expression data such as **maskBAD** (Dannemann et al., 2012) and **SNEP** (Fujisawa et al., 2009). However, models based on two (genetic) groups have limited utility when analyzing more complicated experimental designs such as those found in expression-based analyses using more genetically diverse mouse lines such as Diversity Outbred (Svenson et al., 2012), Collaborative Cross (Collaborative Cross Consortium, 2012) or other Heterogenous Stock (Chia et al., 2005) mice.

In order to facilitate eQTL mapping and other expression analyses in complex mouse crosses we devised an R package **oligoMask** based on the use of high quality publicly available genetic variant databases to screen microarray probes identifying probes impacted by variants. The key to this is the relatively recent availability of genome-wide variant databases in the variant call format (VCF) such as those from the Sanger Mouse Genomes Project (Keane et al., 2011) and the 1000 Genomes for humans (1000 Genomes Project Consortium, 2012) as well as the ability to query and parse these files via the **VariantAnnotation** (Obenchain et al., 2014) package. The **oligoMask** package is designed to work in conjunction with the **oligo** Bioconductor package to facilitate removal of aberrant probe expression prior to the commonly used robust multi-array average (RMA) (Irizarry et al., 2003) pre-processing procedure for Affymetrix arrays. Our package works by removing potentially impacted probes from the overall expression matrix prior to the call to the RMA processing functions. This can be done before the background correction step or after the normalization step. The annotation for these impacted probes are most easily derived from VCF files with the parsed data stored in an SQLite database. This database can be optionally wrapped in an R package with appropriate metadata to facilitate sharing and reproducibility. High-level S4 classes and methods provide a convenient interface with **oligoMask** and **oligo**. In addition, users can define new database schemas, add custom data as well as create their own functionality. Below we give an overview as well as demonstrate using publicly available data the steps involved for the use of **oligoMask**.

Example data

The example data presented in this article and in the vignette was downloaded from the gene expression omnibus (GEO) with accession number GSE33822 (Sun et al., 2012). For demonstration purposes we use a subset (n=8) of the dataset including only those samples derived from whole brain which received the vehicle treatment and that were run on version 1 of the Mouse Gene ST array. In our example, we are looking for expression differences between the NOD/ShiLtJ (NOD) inbred strain and the C57BL/6J (B6) inbred strain, the genome of which serves as the mouse reference genome. As we can expect the microarray probe sequences to be heavily biased towards the reference genome, looking for differential expression between these two strains may be problematic as differences in expression may be due to hybridization artifacts or true gene expression differences. First we create a NOD-specific database and then filter out those probes that are impacted by at least one variant in the NOD strain but not in the B6 strain and then carry out the differential expression analysis as per standard statistical workflows.

Workflow

Creating a variant database

The first step in the use of **oligoMask** is the creation of an SQLite database containing the probe annotation (including alignments to a given genome), variant annotation and the overlap, if any, between probes and variants. In a general sense, the probes sequences are first realigned to the given genome using the **BSgenome** and **Biostrings** Bioconductor packages (Pages, 2013; Pages et al., 2013) with the probe location and mappability of the probes being recorded. The locations of the uniquely mapping probes are then used to compute overlap with the variants in the specified VCF file using import and overlap functionality in the **VariantAnnotation** package. The locations of the variants in the genome, type of variant and the individual/population it was observed in is also recorded along with the overlap between probe alignments and variants. A convenience function for database creation is provided (`create.sanger.mouse.vcf.db`) for use with the case of variants derived from VCF files from the Sanger Mouse Genomes Project and variants of the Affymetrix Mouse Gene ST arrays. The **oligoMask** Vignette demonstrates in the section 'Data preparation' how the NOD variant database package (**om.NOD.mogene.1.0.st**) can be created using this function.

Additional array platforms and variant genotype file types can be supported through a modification of `create.sanger.mouse.vcf.db` as well as specifying the database schema as a "TableSchemaList" object as returned in the pre-defined `SangerTableSchemaList` function. The "TableSchemaList" S4 class serves a similar role as an object-relational mapping approach (ORM) in other languages and allows the R code to interact with a given database in a general way.

Masking procedure

The masking procedure first requires the installation of the **oligo** package along with the appropriate platform design databases that can be downloaded from Bioconductor. In our use case of Affymetrix Gene ST arrays, the CEL files are first read in using the `read.celfiles` function of **oligo** resulting in a "GeneFeatureSet" object. Next, the **oligoMask** database package is loaded, the parameters for the masking procedure are defined and finally the RMA summarization is performed as is shown below starting from the "GeneFeatureSet" object distributed with **oligoMaskData**.

```
library(oligoMask)
library(oligoMaskData)
library(om.NOD.mogene.1.0.st)
library(pd.mogene.1.0.st.v1)
library(limma)
data(oligoMaskData)

var.parms <- VariantMaskParams(om.NOD.mogene.1.0.st, geno.filter = FALSE,
  rm.unmap = FALSE, rm.mult = FALSE)

sun.gfs.mask <- maskRMA(oligoMaskData, target = "core", apply.mask = TRUE,
  mask.params = var.parms)
```

The result of these commands is a summarized "GeneFeatureSet" object with all probes overlapping variants from the NOD inbred strain of mouse removed prior to the background correction step of RMA. Users can control several aspects of the masking procedure through creation of a parameter

object. For instance users can additionally remove probes that map to multiple locations as well as those that do not map at all to the reference genome by supplying `TRUE` to `rm.multi` and/or `rm.unmap`. Similarly, masking can be performed using only those variants that passed quality filters encoded in the VCF file by setting `geno.filter` to `TRUE`.

The `maskRMA` method carries out the RMA procedure and provides a similar interface to the `rma` method from **oligo**. In addition it requires specification of a `"VariantMaskParams"` object and whether the masking procedure should be performed before the background correction function or after background correction and normalization but before summarization by setting the `mask.type` argument to `before.rma` or `before.summary` respectively.

Assessment of masking procedure

As a demonstration of **oligoMask** next we perform a basic linear-model based differential expression analysis with the Sun *et al.* 2012 data comparing results with and without the NOD mask applied. Below we illustrate the basic approach using the masked data.

```
sun.exprs.mask <- exprs(sun.gfs.mask)
phen.dta <- data.frame(t(sapply(strsplit(colnames(sun.exprs.mask), "_"), c))[, 1:3])
names(phen.dta) <- c("tissue", "strain", "exposure")
use.mod <- model.matrix(~strain, data = phen.dta)
fit <- lmFit(sun.exprs.mask, use.mod)
fit <- eBayes(fit)
sun.exprs.mask.res <- decideTests(fit)
```

We then repeat this procedure but this time setting `apply.mask = FALSE` in `maskRMA` to provide the baseline standard RMA values for the comparison.

```
sun.gfs.unmask <- maskRMA(oligoMaskData, target = "core", apply.mask = FALSE,
  mask.params = var.params)

sun.exprs.unmask <- exprs(sun.gfs.unmask)
um.phen.dta <-
  data.frame(t(sapply(strsplit(colnames(sun.exprs.unmask), "_"), c))[, 1:3])
names(um.phen.dta) <- c("tissue", "strain", "exposure")
um.mod <- model.matrix(~strain, data = um.phen.dta)
um.fit <- lmFit(sun.exprs.unmask, um.mod)
um.fit <- eBayes(um.fit)
sun.exprs.unmask.res <- decideTests(um.fit)
```

Finally we produce a summary venn diagram of the results.

```
comb.mat <- cbind(sun.exprs.unmask.res[, "strainNOD", drop = FALSE], Masked = 0)
comb.mat[rownames(sun.exprs.mask.res), "Masked"] <-
  sun.exprs.mask.res[, "strainNOD"]
colnames(comb.mat)[1] <- "UnMasked"
vennDiagram(vennCounts(comb.mat))
```

The Venn diagram provides a visual comparison between the masked and unmasked results, allowing the user to assess impact of variants on expression (Figure 1). An executable version of this code is provided in the accompanying vignette accessed by:

```
Stangle(system.file("doc/oligoMask.Rnw", package = "oligoMask"))
```

Conclusion

oligoMask is a flexible R/SQLite framework for pre-processing and QA/QC of hybridization based expression data. Not only can it remove the effect of spurious probe intensities due to genetic variants but it can additionally correct for design artifacts (probes mapping to multiple places or not mapping at all). It utilizes SQLite and works in conjunction with the **oligo** Bioconductor package. Currently it supports the Affymetrix Mouse Gene ST array though support could be easily added for other array types or species as described above. Approaches for removal of probes based off of the variant and mapping information in the SQLite database are already implemented. More sophisticated algorithms could be built on top of the database to provide masking additionally based off of the

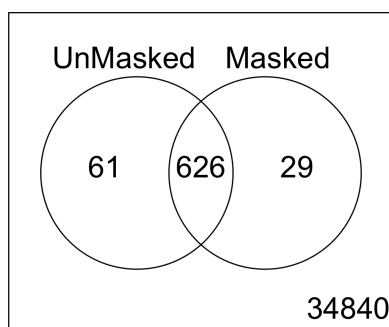


Figure 1: Concordance of differentially expressed genes between the masked and unmasked versions of the analysis

expression data itself or inferred haplotypes. We are currently working on allowing the masking to be further controlled by enforcing positional constraints on the variants relative to the probes as well as enabling masking to be based solely on the mapping information. Source code is freely available to all users from <https://github.com/dbottomly/oligoMask>. Note that **om.NOD.mogene.1.0.st** and **oligoMaskData** are available as part of release 0.99.08 on github (<https://github.com/dbottomly/oligoMask/releases>).

Acknowledgements

We thank the two anonymous reviewers for their comments. This work was supported in part by grants from the National Institute of Allergy and Infectious Disease (1U19AI100625), the National Center for Advancing Translational Sciences (5UL1RR024140) and the National Cancer Institute (5P30CA069533-13).

Bibliography

- 1000 Genomes Project Consortium. An Integrated Map of Genetic Variation From 1,092 Human Genomes. *Nature*, 491:1, 2012. [p159]
- R. Alberts, P. Terpstra, Y. Li, R. Breitling, J.-P. Nap, and R. C. Jansen. Sequence Polymorphisms Cause Many False Cis eQTLs. *PLoS ONE*, 2(7):e622, 2007. [p159]
- D. Benovoy, T. Kwan, and J. Majewski. Effect of Polymorphisms Within Probe-Target Sequences on Oligonucleotide Microarray Experiments. *Nucleic Acids Research*, 36(13):4417–4423, 2008. doi: 10.1093/nar/gkn409. URL <http://nar.oxfordjournals.org/content/36/13/4417.abstract>. [p159]
- B. S. Carvalho and R. A. Irizarry. A Framework for Oligonucleotide Microarray Preprocessing. *Bioinformatics*, 2010. ISSN 1367-4803. doi: <http://dx.doi.org/10.1093/bioinformatics/btq431>. [p159]
- R. Chia, F. Achilli, M. F. Festing, and E. M. Fisher. The Origins and Uses of Mouse Outbred Stocks. *Nature Genetics*, 37(11):1181–1186, 2005. [p159]
- Collaborative Cross Consortium. The Genome Architecture of the Collaborative Cross Mouse Genetic Reference Population. *Genetics*, 190:389–401, 2012. [p159]
- M. Dai, P. Wang, A. D. Boyd, G. Kostov, B. Athey, E. G. Jones, W. E. Bunney, R. M. Myers, T. P. Speed, H. Akil, et al. Evolving Gene/Transcript Definitions Significantly Alter the Interpretation of GeneChip Data. *Nucleic Acids Research*, 33(20):e175–e175, 2005. [p159]
- M. Dannemann, M. Lachmann, and A. Lorenc. ‘maskBAD’ - A Package to Detect and Remove Affymetrix Probes With Binding Affinity Differences. *BMC Bioinformatics*, 13(1):56, 2012. ISSN 1471-2105. doi: 10.1186/1471-2105-13-56. URL <http://www.biomedcentral.com/1471-2105/13/56>. [p159]
- S. Duan, W. Zhang, W. K. Bleibel, N. J. Cox, and M. E. Dolan. SNPInProbe_1.0: A Database for Filtering Out Probes in the Affymetrix GeneChip® Human Exon 1.0 ST Array Potentially Affected by SNPs. *Bioinformatics*, 2(10):469, 2008. [p159]

- H. Fujisawa, Y. Horiuchi, Y. Harushima, T. Takada, S. Eguchi, T. Mochizuki, T. Sakaguchi, T. Shiroishi, and N. Kurata. SNEP: Simultaneous Detection of Nucleotide and Expression Polymorphisms Using Affymetrix GeneChip. *BMC Bioinformatics*, 10(1):131, 2009. ISSN 1471-2105. doi: 10.1186/1471-2105-10-131. URL <http://www.biomedcentral.com/1471-2105/10/131>. [p159]
- R. A. Irizarry, B. Hobbs, F. Collin, Y. D. Beazer-Barclay, K. J. Antonellis, U. Scherf, and T. P. Speed. Exploration, Normalization, and Summaries of High Density Oligonucleotide Array Probe Level Data. *Biostatistics*, 4(2):249–264, 2003. [p159]
- T. M. Keane, L. Goodstadt, P. Danecek, M. A. White, K. Wong, B. Yalcin, A. Heger, A. Agam, G. Slater, M. Goodson, et al. Mouse Genomic Variation and Its Effect on Phenotypes and Gene Regulation. *Nature*, 477(7364):289–294, 2011. [p159]
- S. Kumari, L. Verma, and J. Weller. AffyMAPSDetector: A Software Tool to Characterize Affymetrix GeneChip® Expression Arrays With Respect to SNPs. *BMC Bioinformatics*, 8(1):276, 2007. ISSN 1471-2105. doi: 10.1186/1471-2105-8-276. URL <http://www.biomedcentral.com/1471-2105/8/276>. [p159]
- V. Obenchain, M. Lawrence, V. Carey, S. Gogarten, P. Shannon, and M. Morgan. VariantAnnotation: A Bioconductor Package for Exploration and Annotation of Genetic Variants. *Bioinformatics*, 2014. doi: 10.1093/bioinformatics/btu168. URL <http://bioinformatics.oxfordjournals.org/content/early/2014/03/28/bioinformatics.btu168.abstract>. [p159]
- H. Pages. *BSgenome: Infrastructure for Biostrings-Based Genome Data Packages*, 2013. R package version 1.30.0. [p160]
- H. Pages, P. Aboyoun, R. Gentleman, and S. DebRoy. *Biostrings: String Objects Representing Biological Sequences, and Matching Algorithms*, 2013. R package version 2.30.0. [p160]
- C. Stratowa, Vienna, and Austria. *xps: Processing and Analysis of Affymetrix Oligonucleotide Arrays including Exon Arrays, Whole Genome Arrays and Plate Arrays*, 2013. [p159]
- W. Sun, S. Lee, V. Zhabotynsky, F. Zou, F. A. Wright, J. J. Crowley, Z. Yun, R. J. Buus, D. R. Miller, J. Wang, et al. Transcriptome Atlases of Mouse Brain Reveals Differential Expression Across Brain Regions and Genetic Backgrounds. *G3: Genes | Genomes | Genetics*, 2(2):203–211, 2012. [p160]
- K. L. Svenson, D. M. Gatti, W. Valdar, C. E. Welsh, R. Cheng, E. J. Chesler, A. A. Palmer, L. McMillan, and G. A. Churchill. High-Resolution Genetic Mapping Using the Mouse Diversity Outbred Population. *Genetics*, 190(2):437–447, 2012. [p159]
- N. A. Walter, S. K. McWeeney, S. T. Peters, J. K. Belknap, R. Hitzemann, and K. J. Buck. SNPs Matter: Impact on Detection of Differential Expression. *Nature Methods*, 4(9):679–680, 2007. [p159]

Daniel Bottomly
Oregon Clinical and Translational Research Institute
Oregon Health and Science University
3181 SW Sam Jackson Park Rd.
Portland, Oregon 97239
USA bottomly@ohsu.edu

Beth Wilmot
Oregon Clinical and Translational Research Institute
Division of Bioinformatics and Computational Biology, DMICE
3181 SW Sam Jackson Park Rd.
Portland, Oregon 97239
USA wilmotb@ohsu.edu

Shannon K. McWeeney
Oregon Clinical and Translational Research Institute
Knight Cancer Institute
Division of Bioinformatics and Computational Biology, DMICE
Division of Biostatistics, PHPM
3181 SW Sam Jackson Park Rd.
Portland, Oregon 97239
USA mcweeney@ohsu.edu

sgr: A Package for Simulating Conditional Fake Ordinal Data

by Luigi Lombardi and Massimiliano Pastore

Abstract Many self-report measures of attitudes, beliefs, personality, and pathology include items that can be easily manipulated by respondents. For example, an individual may deliberately attempt to manipulate or distort responses to simulate grossly exaggerated physical or psychological symptoms in order to reach specific goals such as, for example, obtaining financial compensation, avoiding being charged with a crime, avoiding military duty, or obtaining drugs. This article introduces the package **sgr** that can be used to perform fake data analysis according to the sample generation by replacement approach. The package includes functions for making simple inferences about discrete/ordinal fake data. The package allows to quantify uncertainty in inferences based on possible fake data as well as to study the implications of fake data for empirical results.

Introduction

How can we evaluate the impact of fake information in real life contexts? In nature, some individuals tend to distort their behaviors or actions in order to reach specific goals. In some species, for example, wimpy animals may not signal their real social value by faking a higher status to deceive other competitors. Similarly, in personnel selection some job applicants may misrepresent themselves on a personality test hoping to increase the likelihood of being offered a job. Being able to discriminate between honest and fraudulent signals and evaluating the impact of counterfeit elements crucially depend on the way we can reason about the whole process of faking. A coherent knowledge of the type or structure of faking processes may lead to stronger inferences that lie on or close to what we may call the genuine, but probably hidden, representation of a manifest behavior. In general fake data may alter a large variety of self-report measures. This problem is particularly relevant for discrete/ordinal data collected in sensitive environments such as, for example, risky sexual behaviors, drug addictions, tax evasion, political preferences, financial compensation, and personnel selection. More in general, researchers interested in the study of human behavior in areas like psychology (Hopwood et al., 2006), organizational and social science (Van der Geest and Sarkodie, 1998), psychiatry (Beaber et al., 1985), forensic medicine (Gray et al., 2003), scientific frauds (Marshall, 2000), and economics (Crawford, 2003) may face the fake data problem when analyzing and interpreting empirical data.

In this article, we discuss the **sgr** package that we have developed for running fake data analysis according to the *sample generation by replacement* (SGR) approach (Lombardi and Pastore, 2012). SGR is a data simulation procedure to generate artificial samples of fake discrete/ordinal data. The main characteristic of the SGR approach is that it allows detailed explorations of what outcomes are produced by particular sets of faking assumptions. By changing the input in the faking model parameters and showing the effect on the outcome of a model, SGR provides a *what-if-analysis* of the faking scenarios. Therefore, SGR can be used to quantify uncertainty in inferences based on possible fake data as well as to evaluate the implications of fake data for statistical results. To illustrate, let us consider the following example where a researcher is interested in studying the relationship between therapy-uncompliance indicators (e.g., forgetting the treatment) and unsafe behaviors indicators (e.g., drinking alcohol) in a group of liver transplant patients. Generally, patients diagnosed with alcohol dependence who follow a pharmaceutical regimen after the liver transplant would deliberately answer fraudulently a question about drinking alcohol due to abstinence from ethanol and social desirability factors (e.g. Foster et al., 1997). In this context, an SGR analysis can help in testing for potential influence of faking the drinking alcohol self-report measure on the strength of the relationship between therapy-uncompliance and unsafe behaviors indicators. More specifically, how sensitive are the empirical associations to possible fake observations in the drinking alcohol self-report measure? Are the conclusions still valid under one or more scenarios of faking (e.g., slight, moderate, and extreme faking) for the drinking alcohol variable?

In general, SGR takes an interpretation perspective by incorporating in a global model all the available information about the process of faking and the underlying true model representation. This makes SGR related in spirit to other statistical approaches such as, for example, uncertainty and sensitivity analysis (Helton et al., 2006) and prospective power analysis (Cohen, 1988) which are all characterized by an attempt to directly quantify uncertainty of general statistics computed on the data by means of specific hypotheses.

The rest of the paper is organized as follows. The next section reviews the SGR framework and its basic implementations using the **sgr** package. The following section provides three examples

illustrating the application of **sgf** to faking scenarios. The final section discusses limitations and future implementations in **sgf** components beyond the general scheme presented here.

The SGR framework

SGR is characterized by a two-stage sampling procedure based on two distinct generative models: the model defining the process that generates the data prior to any fake perturbation (*data generation process*) and the model representing the faking process to perturb the data (*data replacement process*). By repeatedly sampling data from the SGR procedure we can generate the so called fake data sample (FDS) and eventually study the distribution of some relevant statistics computed on these simulated data samples. In SGR the data generation process is modeled by means of standard Monte Carlo procedures for ordinal data whereas the data replacement process is implemented using ad hoc probabilistic faking models. In sum, the overall generative process is split into two conceptually independent and possibly simpler components (divide and conquer strategy).

With regard to the fake-data problem in general, we think of the data in the generation process as being represented by an $n \times m$ matrix \mathbf{D} , that is to say, n i.i.d. observations (hypothetical participants) each containing m elements (hypothetical participant's responses). We assume that entry d_{ij} of \mathbf{D} ($i = 1, \dots, n; j = 1, \dots, m$) takes values on a small ordinal range $V_q = \{1, \dots, q\}$ (for the sake of simplicity, in this presentation we assume identical ordinal scales). In particular, let \mathbf{d}_i be the $(1 \times m)$ array of \mathbf{D} denoting the pattern of responses of participant i . The response pattern \mathbf{d}_i is a multidimensional ordinal random variable with probability distribution $p(\mathbf{d}_i|\theta_D)$, where θ_D indicates the vector of parameters of the probabilistic model for the data generation process. The main idea of the replacement approach is to construct a new $n \times m$ ordinal data matrix \mathbf{F} , called the *fake data matrix* of \mathbf{D} , by manipulating each element d_{ij} in \mathbf{D} according to a replacement probability distribution. Let \mathbf{f}_i be the $(1 \times m)$ array of \mathbf{F} denoting the replaced pattern of fake responses of participant i . The fake response pattern \mathbf{f}_i is a multidimensional ordinal random variable with conditional replacement probability distribution

$$p(\mathbf{f}_i|\mathbf{d}_i, \theta_F) = \prod_{j=1}^m p(f_{ij}|d_{ij}, \theta_F), \quad i = 1, \dots, n \quad (1)$$

where θ_F indicates the vector of parameters of the probabilistic faking model.

It is important to note that in the standard SGR framework the replacement distribution $p(\mathbf{f}_i|\mathbf{d}_i, \theta_F)$ is restricted to satisfy the *conditional independence* (CI) assumption (see Lombardi and Pastore, 2012; Pastore and Lombardi, 2014). More precisely, in the replacement distribution each fake response f_{ij} only depends on the corresponding data observation d_{ij} and the model parameter θ_F . Therefore, because the patterns of fake responses are also i.i.d. observations, the simulated data array (\mathbf{D}, \mathbf{F}) is drawn from the joint probability distribution

$$p(\mathbf{D}, \mathbf{F}|\theta_D, \theta_F) = \prod_{i=1}^n p(\mathbf{d}_i|\theta_D) p(\mathbf{f}_i|\mathbf{d}_i, \theta_F) \quad (2)$$

$$= \prod_{i=1}^n p(\mathbf{d}_i|\theta_D) \prod_{j=1}^m p(f_{ij}|d_{ij}, \theta_F) \quad (3)$$

In the last section of this article we will discuss some potential limitations of the conditional independence assumption in real application domains of the SGR approach.

Data generation process

In general, several options are available to represent the data generation process (Muthén, 1984; Jöreskog and Sörbom, 1996; Moustaki and Knott, 2000; Samejima, 1969). In the current version of the **sgf** package we implemented a procedure based on the multivariate latent variable framework which is called *underlying variable approach* (UVA, Muthén, 1984; Jöreskog and Sörbom, 1996). This approach assumes that the observed ordinal variables are treated as metric through assumed underlying normal variables. In particular, we assume that there exists a continuous data matrix \mathbf{D}^* underlying the ordinal data matrix \mathbf{D} . Let \mathbf{d}_i^* be the $(1 \times m)$ array of \mathbf{D}^* denoting the pattern of underlying continuous values of the i th observation. It is convenient to let \mathbf{d}_i^* have the multivariate standard normal distribution with density function $\phi(\mathbf{0}, \mathbf{R})$ where \mathbf{R} denotes the $(m \times m)$ model correlation matrix. The connection between the ordinal variable d_{ij} and the underlying variable d_{ij}^* in \mathbf{D}^* is given by

$$d_{ij} = h \quad \text{iff} \quad \tau_{h-1}^j < d_{ij}^* \leq \tau_h^j$$

with $h = 1, \dots, q; i = 1, \dots, n; j = 1, \dots, m$ and where

$$-\infty = \tau_0^j < \tau_1^j < \tau_2^j < \dots < \tau_{q-1}^j, \tau_q^j = +\infty,$$

are threshold parameters. Therefore, the joint probability of $\mathbf{d}_i = (h_1, \dots, h_m)$ is given by

$$p(\mathbf{h}|\theta_M) = \int_{\tau_{h_1-1}^1}^{\tau_{h_1}^1} \dots \int_{\tau_{h_m-1}^m}^{\tau_{h_m}^m} \phi(\mathbf{z}|\mathbf{0}, \mathbf{R}) d\mathbf{z} \quad (4)$$

with $\theta_M = (\tau, \mathbf{R})$ and $\mathbf{z} = (z_1, \dots, z_m)$ being the parameter vector of the original data generation model and the values for the continuous variables, respectively.

In SGR the data generation process is obtained by first generating the continuous data \mathbf{D}^* according to a model correlation matrix \mathbf{R} and then by transforming it to its discrete counterpart \mathbf{D} using the model thresholds τ . In the following example, we used the `sgr` function `rdatagen` to sample $n = 100$ random observations from a data generation model with two symmetrically distributed ordinal variables with five levels each and correlation value .4.

```
> library(sgr)
> require(MASS)
> require(polycor)
> set.seed(367)
> R <- matrix(c(1, .4, .4, 1), 2, 2)
> th <- list(c(-Inf, qnorm(c(0.04, 0.27, 0.73, 0.96))), Inf),
+          c(-Inf, qnorm(c(0.06, 0.31, 0.69, 0.94))), Inf))
> Dx <- rdatagen(n=100, R=R, Q=c(5, 5), th=th)
> Dx$data
```

In this example, the threshold values are derived from the quantiles of the standard normal distribution in such a way that the first simulated variable shows a slightly larger variance than the second simulated variable. Generally, the threshold values can be derived in two different ways. In the first case, we can use empirically based knowledge (e.g., an already existing data set) to estimate the threshold values on the basis of the observed distribution function of the levels of the discrete variable (e.g., Jöreskog and Sörbom, 1996). In the second case, some simple statistical knowledge can be used to simulate threshold values according to desired properties. For example, the normal quantiles used as corresponding threshold values can be computed using the inverse of the binomial cumulative distribution function (e.g., Jöreskog and Sörbom, 1996). In the `rdatagen` function call the parameter `Q` specifies the number of levels for each ordinal variable. To compare the model correlation matrix \mathbf{R} with the sample polychoric correlation, we can use the `polychor` function in the `polycor` package (Fox, 2010)

```
> d1 <- factor(Dx$data[, 1], ordered=TRUE)
> d2 <- factor(Dx$data[, 2], ordered=TRUE)
> polychor(d1, d2, ML=TRUE, std.err=TRUE)
```

Polychoric Correlation, ML est. = 0.3627 (0.09832)

Data replacement process

To generate the fake ordinal data we used a parametrized replacement distribution based on a discrete beta kernel (Pastore and Lombardi, 2014). Some examples of replacement distributions are shown in Figure 1. Let $p_{k|h} \equiv p(k|h, \theta_F)$ be the conditional probability of replacing an original ordinal value h with the new ordinal value k . In general, θ_F represents hypothetical a priori knowledge about the distribution of faking (e.g., the chance of observing a fake observation in the data) or empirically based knowledge about the process of faking (e.g., the direction of faking -fake good vs fake bad-).

The conditional replacement distribution can be described according to the following equation

$$p_{k|h} = \begin{cases} DG(k; a^+, b^+, \theta_F^+) \pi^+, & 1 \leq h < k \leq q \\ DG(k; a^-, b^-, \theta_F^-) \pi^-, & 1 \leq k < h \leq q \\ 1 - (\pi^+ + \pi^-), & 1 < k = h < q \\ 1 - \pi^+, & k = h = 1 \\ 1 - \pi^-, & k = h = q \end{cases} \quad (5)$$

with DG being the generalized beta distribution for discrete variables (Pastore and Lombardi, 2014). Note that in Eq. (5), the function DG is used with two different set of parameters. More precisely, in the first line the function DG models the behavior of the faking distribution for fake positive values

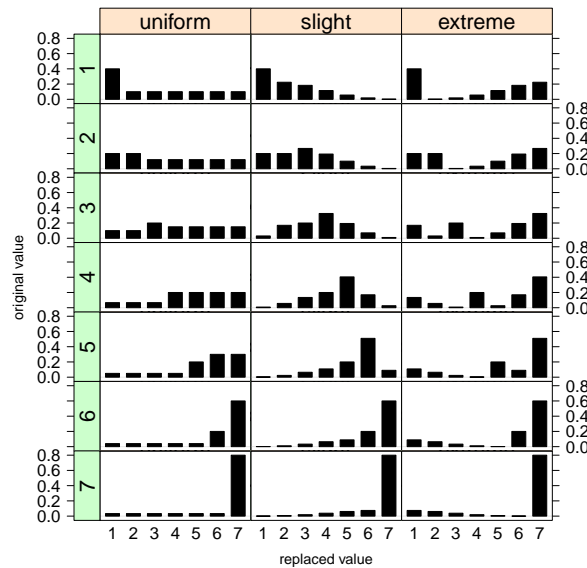


Figure 1: Three examples of conditional replacement distributions for a 7-point discrete r.v. Each column in the graphical representation corresponds to a different conditional replacement distribution (see Table 1). For each example the overall probabilities are $\pi^+ = .6$ and $\pi^- = .2$. Each row in the graphical representation corresponds to a different original 7-point discrete value h .

($k > h$) by means of the governing shape parameter $\theta_F^+ = (\gamma^+, \delta^+)$ with bounds ($a^+ = h + 1, b^+ = q$). By contrast, the second line represents the behavior of the faking distribution for fake negative values ($k < h$) modelled by the governing shape parameter $\theta_F^- = (\gamma^-, \delta^-)$ with bounds ($a^- = 1, b^- = h - 1$). Some examples of faking models with their parameters assignments are reported in Table 1 (see also Pastore and Lombardi, 2014). In general, the governing shape parameters θ_F^+ and θ_F^- must be strictly positive. In particular, if $\gamma^+ = \delta^+ = 1$, the right part of the replacement distribution reduces to a *uniform* support fake positive distribution (Fig. 1 first column). By contrast, if $1 \leq \gamma^+ < \delta^+$ (resp. $1 \leq \delta^+ < \gamma^+$), the model mimics asymmetric faking configurations corresponding to moderate positive shifts (resp. exaggerated positive shifts) in the value of the original response (Fig. 1, second and third columns). More specifically, in the *slight* positive faking configuration the chance to replace an original value h with another greater value k decreases as a function of the distance between k and h . By contrast, in the *extreme* positive faking configuration the chance to replace an original value h with another greater value k increases as a function of the distance between k and h . Unlike the asymmetric configurations (slight faking and extreme faking), the uniform support distribution ($\gamma^+ = \delta^+ = 1$) mimics a kind of random world model that can be used whenever we believe to deal with purely random fake data. This principle requires the simplest quantitative representation for the replacement process and reflects the lack of information about the distributional properties of the faking behavior. Similar configurations can be described also for the left part of the replacement distribution which represents the negative faking process [$\theta_F^- = (\gamma^-, \delta^-)$]. However, for this latter component the ordinal relation characterizing the shape parameters must be reversed (see Table 1). Finally, in the conditional replacement distribution the parameters π^+ and π^- denote the overall probability of faking positive and the overall probability of faking negative, respectively. These probabilities act as weights to rescale the discrete beta distribution DG such that $(\pi = \pi^+ + \pi^-) \leq 1$. In general, π^+ and π^- represent *a priori* or empirically based knowledge about the distribution of faking for the two components (e.g., the chance of observing a positive or negative fake observation in the data). The third, fourth, and fifth lines of Eq. (5) show the probability of non-replacement ($k = h$). Note that, if we set $\pi^+ = 0$ (resp. $\pi^- = 0$), then the replacement model boils down to a pure faking negative (resp. positive) model which corresponds to a context in which responses are exclusively subject to negative (resp. positive) faking (see fig. 2).

In the following example, we applied a pure (slight) positive faking model (see Table 1) to generate a fake data matrix F from the original data matrix D .

```
> RM <- replacement.matrix(Q=5,p=c(.5,0),fake.model="slight")
> Fx <- rdatarepl(Dx$data,RM)
46% of data replaced.
> Fx$Fx
```

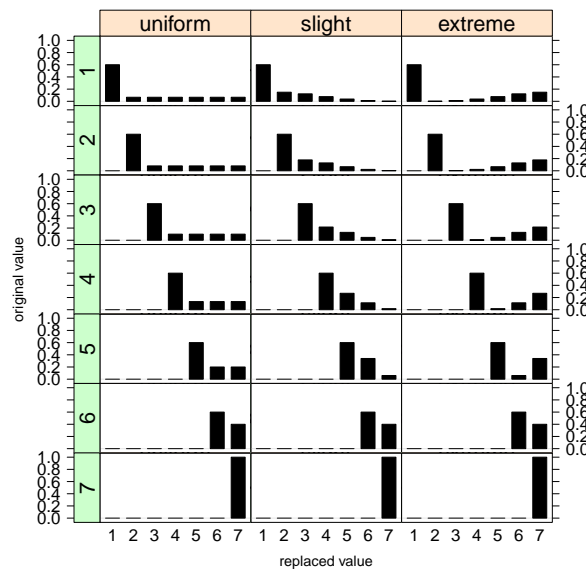


Figure 2: Three examples of conditional replacement distributions for a 7-point discrete r.v. Each column in the graphical representation corresponds to a different conditional replacement distribution. For each example the overall probabilities are $\pi^+ = .6$ and $\pi^- = .0$ (faking positive condition). Each row in the graphical representation corresponds to a different original 7-point discrete value h .

We used the `replacement.matrix` function to construct the conditional replacement probability distribution and save the result in the variable `RM` which is used as the argument of the data replacement generation function `rdaterepl`. Note that the argument `fake.model` in the `replacement.matrix` function allows to set the options reported in Table 1. However, all of the model parameters can be set manually by the user to any array of consistent values if so desired. For example, an equivalent syntax would have been

```
> RM <- replacement.matrix(Q=5, p=c(.5, 0), gam=c(1.5, 0), del=c(4, 0))
```

We can evaluate the impact of positive faking on the new fake data matrix by comparing the frequencies of the ordinal categories in `D` and `F`. For example, for the first ordinal variable we have

```
> table(Dx$data[, 1])
```

```
1  2  3  4  5
5 29 40 24  2
```

```
> table(Fx$Fx[, 1])
```

```
1  2  3  4  5
2 17 36 31 14
```

which shows how the positive faking has shifted the values of the first ordinal variable towards larger ones. In a similar way, we could also evaluate the impact of faking on the sample polychoric correlation matrix of `F`.

Model	γ^+	γ^-	δ^+	δ^-
uniform	1	1	1	1
slight	1.5	4	4	1.5
extreme	4	1.5	1.5	4

Table 1: Examples of default parameters assignments for some relevant faking models (Pastore and Lombardi, 2014).

Illustrative examples

By way of illustration we consider three simple SGR examples. The first is for the evaluation of a correlational analysis computed on five-point rating data. This example is hypothetical and serves to introduce the main features and functions implemented in the **sgr** package. The second application considers real data about illicit drug use among young people aged 14 to 27. This second example shows how to model directional faking hypotheses (e.g., faking good or faking bad). It is also important because illustrates how the replacement functions can be applied to dichotomous data. Finally, the third application extends the second example by analyzing a new set of data about cannabis consumption in young people using log-linear models for ordinal data.

Example 1

We begin with a simple SGR analysis about a hypothetical observed difference ($\hat{\Delta} = \hat{\rho}_1 - \hat{\rho}_2 = .3$) between two ordinal correlations computed on two five-point rating variables X and Y for the groups of subjects, G_1 ($n_1 = 50$) and G_2 ($n_2 = 50$). For example, in a risky sexual behaviors scenario the rating variables X and Y can represent, in two groups of young adults (females and males), the self-report attitude to contraceptive use during a sexual intercourse and the declared number of sexual partners in the last three months, respectively. Normally, an effect size of .3 denotes a relevant difference between two correlations. However, how sensitive may this result be to possible fake data? Is this effect still observed under one or more scenarios of faking? In this example, we are interested in testing whether the observed correlation difference can still be consistent with a true generative model reflecting an identical moderate correlation $\rho_1 = \rho_2 = .25$ for the two groups. Moreover, we also assume a perturbation process represented by two distinct uniform faking models: $\pi_1^+ = .2$ and $\pi_1^- = .1$ for G_1 , and $\pi_2^+ = .3$ and $\pi_2^- = .2$ for G_2 . We can easily reformulate this example using a Fisher significance testing (Lehmann, 1993). More precisely, we can construct the corresponding hypothesis

$$\begin{aligned} H : \quad & \rho_1 = \rho_2 = .25 \ (\Delta = 0), \\ & \pi_1^+ = \pi_2^- = .2, \pi_1^- = .1, \pi_2^+ = .3, \\ & \gamma_s^+ = \gamma_s^- = \delta_s^+ = \delta_s^- = 1, \quad s = 1, 2 \end{aligned}$$

and examine whether or not the observed correlation difference $\hat{\Delta}$ is consistent with H . In particular, we are interested in the p -value

$$Pr[\Delta > \hat{\Delta} | H].$$

The code below illustrates the SGR analysis

```
> require(polycor)
> set.seed(367)
> obs.stat <- .3; mc.stat <- NULL
> Rmc <- matrix(c(1,.25,.25,1),2)
> PM <- matrix(c(rep(1,100),rep(2,100)),ncol=2,byrow=TRUE)
> Pparm <- list(p=matrix(c(.2,.3,.1,.2),2),gam=matrix(1,2,2),del=matrix(1,2,2))
> for (b in 1:1000) {
+   mcD <- rdatagen(n=100,R=Rmc,Q=5)$data
+   Fx <- partition.replacement(mcD,PM,Pparm=Pparm)
+   for (j in 1:ncol(Fx)) {
+     Fx[,j] <- ordered(Fx[,j])
+   }
+   mcpc1 <- hetcor(Fx[1:50,])$correlations[1,2]
+   mcpc2 <- hetcor(Fx[51:100,])$correlations[1,2]
+   Delta <- mcpc1-mcpc2
+   mc.stat <- c(mc.stat,Delta)
+ }

> hist(mc.stat)
> sum(mc.stat>=obs.stat)/1000
[1] 0.226
```

An empirical p -value can be computed by a Monte Carlo experiment. In our example, the test procedure $\Delta^* = \hat{\rho}_1^* - \hat{\rho}_2^*$ is replicated 1000 times under the condition of the hypothesis. Next, the approximate p -value is computed as the proportion of the simulated Δ^* values which are larger than the observed correlation difference .3. More precisely, for each replicate $b = 1, \dots, 1000$, we first generate a 100×2 ordinal data matrix mcD according to the generative model with correlation

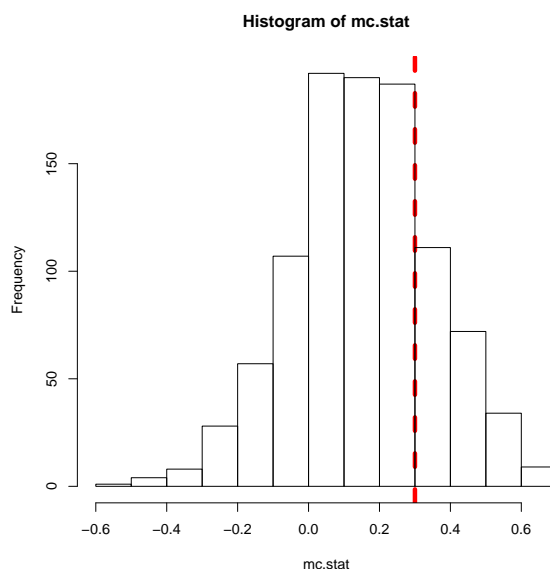


Figure 3: Distribution of the test procedure Δ^* under H .

matrix `Rmc`. This matrix contains two symmetrically distributed ordinal variables (default value¹ in the `rdatagen` function.) Next, the ordinal matrix is transformed according to the faking models. In particular, the function `partition.replacement` allows to set different replacement distributions for the two groups of subjects and returns the perturbed data matrix. This function has three main arguments: `Dx=mcD`, the data frame or matrix to be replaced; `PM`, the partition matrix to cluster the observations into the distinct groups; `Pparm`, the list of replacements parameters for each of the different faking models. Note that the partition matrix must have the same dimension as the matrix to be replaced and a numeric code for each distinct cluster (group) in the partition. If a cell of the partition matrix contains 0, then the corresponding cell value in the original data matrix is not modified (*no replacement* condition is applied). In our example, `Pparm` is a list containing three elements. Each element is a 2 (number of groups) \times 2 (faking positive vs faking negative) matrix. So for example, `p[1, 1]` and `p[1, 2]` denote the overall faking positive probabilities for G_1 and G_2 , respectively. Similarly, `gam[1, 1]` (resp. `gam[2, 1]`) indicates the first shape parameter for the faking positive (resp. faking negative) model in group G_1 . The same figure follows for the second shape parameter `de1`. Figure 3 shows the distribution of the test procedure under H (approximate p -value = .226). According to the distribution of the test procedure the observed correlation difference $\hat{\Delta}$ seems consistent with the hypothesis of faking.

Example 2

Table 2 refers to a real prospective study about illicit drug use among young people aged 14-27 (Pastore et al., 2007). In particular, we evaluated the relationship between age (dichotomized into two categories: adults, > 17 , and minors) and ecstasy drug consumption. We expected that each individual would deliberately answer the question either honestly or fraudulently depending on her/his beliefs and intentions which, in turn, could be influenced by the context. How can the researcher evaluate the impact of possible fake answers when trying to provide an overall picture of the investigated phenomena? Although the example is specific, a similar problem may occur in a variety of situations about stigmatizing characteristics (e.g., habitual gambling, experience of induced abortion, tax evasion, rash driving, risky sexual behavior).

The result of a log linear model for independence for the two-way table showed a significant likelihood-ratio chi-squared statistic ($G^2_{(1)} = 5.29, p < .05$). Hence the independence assumption was rejected. By a quick inspection of the counts shown in table 2 we can easily recognize that only 29% of adults answered affirmatively to the question. By contrast, more than 50% of minors replied affirmatively. Therefore, we suspected that the adults might have shown a larger level of

¹The default setting requires that the quantiles are computed using the inverse of the binomial cumulative distribution (see for example, Jöreskog and Sörbom, 1996).

	drug	
	yes (1)	no (2)
adults (1)	10	25
minors (2)	32	29

Table 2: Observed frequencies for testing independence of age and drug use for the question: Have you ever made use of ecstasy?

social desirability (Paulhus, 1984) as compared to the minors. This might have artificially boosted the observed difference between the two groups. To test this hypothesis we performed a new SGR analysis on the two-way table by assuming a) a generative model implementing the independence assumption with marginal probability $\Pr(\text{yes}) = .75$ b) a fake good model for the variable drug consumption. In general, faking good can be conceptualized as an individual's deliberate attempt to manipulate or distort responses to create a positive impression (Paulhus, 1984). Notice that, the faking good (as well as the faking bad) scenario always entails a conditional replacement model in which the conditioning is a function of response polarity. In this application the scenario corresponds to a context in which all fakers respond negatively to the question. Finally, we also assumed two distinct levels of faking for the two groups: $\pi_1^+ = .8$ for the adults and $\pi_2^+ = .4$ for the minors. We reformulated the problem within a pure significance test setting:

$$\begin{aligned}
 H &: G^2 = 0 \text{ (independence assumption),} \\
 &\Pr(\text{yes}) = .75, \\
 &\pi_1^+ = .8, \pi_2^+ = .4, \pi_1^- = \pi_2^- = .0, \\
 &\gamma_s^+ = \delta_s^+ = 1, \gamma_s^- = \delta_s^- = 0, \quad s = 1, 2
 \end{aligned}$$

The following code illustrates the SGR analysis

```

> require(MASS)
> set.seed(367)
> data(smokers)
> ecstasy.table <- table(smokers$drug, smokers$age, dnn=c("drug", "age"))
> obs.lrt <- loglm(~drug+age, data=ecstasy.table)$lrt
>
> PM <- matrix(0, nrow(smokers), 2)
> PM[smokers$age==1, 2] <- 1
> PM[smokers$age==2, 2] <- 2
> Pparm <- list(p=matrix(c(.8, .4, 0, 0), 2), gam=matrix(c(1, 1, 0, 0), 2),
+             del=matrix(c(1, 1, 0, 0), 2))
> mc.lrt <- NULL
> for (b in 1:1000) {
+   smokers$simdrug <- rdatagen(nrow(smokers), R=matrix(1), Q=2,
+                               probs=list(c(.75, .25)))$data
+   Fx <- partition.replacement(smokers[, c("age", "simdrug")], PM, Pparm=Pparm)
+   mc.lrt <- c(mc.lrt, loglm(~simdrug+age, data=table(Fx$simdrug, Fx$age,
+                                                       dnn=c("simdrug", "age")))$lrt)
+ }

> hist(mc.lrt)
> sum(mc.lrt >= obs.lrt) / 1000
[1] 0.812

```

Note that for dichotomous variables ($q = 2$) all faking positive models reduce to the following uniform conditional replacement distribution (Pastore and Lombardi, 2014).

$$p_{k|h} = \begin{cases} 1, & h = k = 2 \\ \pi^+, & h = 1, k = 2 \\ 1 - \pi^+, & h = k = 1 \\ 0, & h = 2, k = 1 \end{cases} \quad (6)$$

Figure 4 shows the distribution of the test procedure under the hypothesis (approximate p -value = .812). According to the approximate G^2 distribution the observed likelihood ratio seems consistent with the hypothesis of faking.

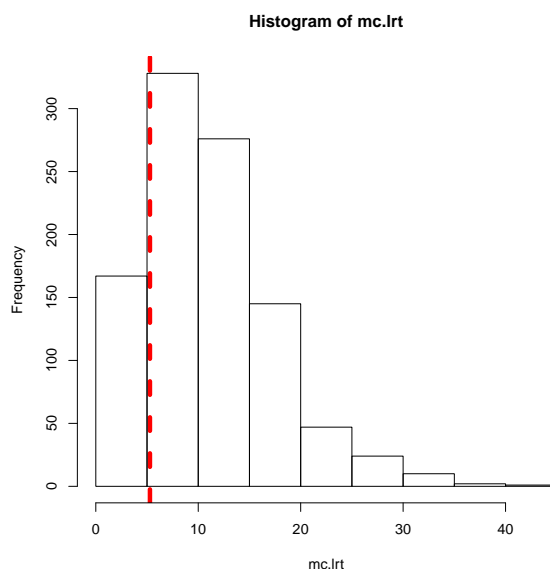


Figure 4: Reproduced distribution for the test statistic G^2 under H .

Example 3

In this application we extend the results reported in the second example by analyzing a new set of ordinal data about illicit drug use among young people (see table 3). This new two-way table relates an independent categorical variable, *age*, minors (< 18 years old) vs adults, to a dependent ordinal variable, *cannabis consumption*. In particular, the dependent variable uses a four-point ordinal scale ranging from *never* (1) to *often* (4) (with intermediate levels being *once* (2) and *some times* (3), respectively). When response categories are ordered, logit models can directly incorporate the ordering (Agresti, 1990). In general, this results in model representations having simpler interpretations than ordinary multicategory logit models at least when the proportional odds model holds.

	cannabis			
	(1)	(2)	(3)	(4)
adults (1)	20	5	7	0
minors (2)	27	5	18	10

Table 3: Observed frequencies for testing independence of age and drug use.

The following code illustrates the results of applying an ordered logistic model to the data represented in table 3. For the analysis we used the function `polr` in the **MASS** package (Venables and Ripley, 2002) that allows to fit a logistic or probit regression model to an ordered factor response.

```
> Y <- data.frame(list(age=gl(2,4),response=gl(4,1,8,ordered=TRUE),
+ counts=c(20,5,7,0,27,5,18,10)))
> fit0 <- polr(response~1,data=Y,weight=counts)
> fit1 <- polr(response~age,data=Y,weight=counts)
> lrt.obs <- -2*(logLik(fit0)-logLik(fit1))
```

The likelihood ratio statistic $\Lambda = -2(L_0 - L_1)$ for the observed sample showed a significant result ($\Lambda_{(3)} = 5.22, p < .05$). Hence, the independence assumption was rejected in the logit model. About the model of faking also in this application we expected that individuals' responses were strictly subject to faking good manipulations. However, unlike the previous example, this time we speculated that only the group of adults showed a social desirability bias whereas the minors' responses were assumed not to be fake dependent ($\pi_2 = \pi_2^+ + \pi_2^- = 0$). In particular, we supposed that the adults were showing a moderate level of faking good (10%) and that their responses were characterized by a slight faking behavior (see table 1). Note that because of the meaning of the categories of the ordinal scale for cannabis consumption, in this application the faking good manipulations are modelled by means of the fake negative parameters (π^-). Finally, for the data generation process we constructed a

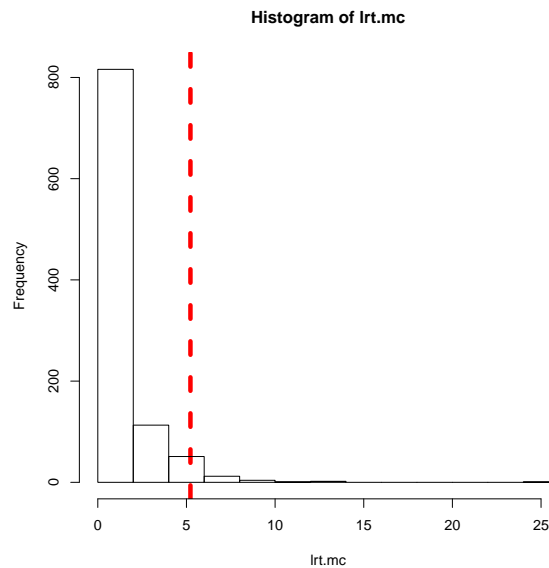


Figure 5: Reproduced distribution for the likelihood ratio statistic under H .

generative model under the assumption of no relation between age and cannabis consumption ($\Lambda = 0$) and with true response proportions being equal to the empirical response proportions for the group of minors. We can collect all the information in the following hypothesis:

$$\begin{aligned}
 H : \quad & \Lambda = 0 \text{ (independence assumption),} \\
 & \Pr(0) = .45, \Pr(1) = .08, \\
 & \Pr(2) = .30, \Pr(3) = .17, \\
 & \pi_1^- = .1, \pi_2^- = \pi_1^+ = \pi_2^+ = .0, \\
 & \gamma_1^- = 1.5, \delta_1^- = 4, \\
 & \text{all the other shape parameters are set to 0}
 \end{aligned}$$

The following code illustrates the SGR analysis

```

> set.seed(367)
> Z <- na.omit(smokers[,c("age", "druguse")])
> PM <- matrix(0, nrow(Z), ncol(Z))
> PM[Z$age==1, 2] <- 1
> lrt.mc <- NULL
> for (b in 1:1000) {
+   Z$simdrug <- rdatagen(nrow(Z), R=matrix(1), Q=4,
+                         probs=list(c(27, 5, 18, 10)/60))$data
+   Dx <- Z[, -2]
+   Fx <- partition.replacement(Dx, PM, p=matrix(c(0, .1), 1), fake.model="slight")
+   Tmc <- table(Fx$age, Fx$simdrug)
+   Ymc <- data.frame(list(age=gl(2, 4), response=gl(4, 1, 8, ordered=TRUE),
+                           counts=c(Tmc[1, ], Tmc[2, ])))
+   fit0 <- polr(response~1, data=Ymc, weight=counts)
+   fit1 <- polr(response~age, data=Ymc, weight=counts)
+   lrt.mc <- c(lrt.mc, -2*(logLik(fit0)-logLik(fit1)))
+ }

> sum(lrt.mc >= lrt.obs)/1000
[1] 0.039

```

Figure 5 shows the distribution of the test procedure under the hypothesis. This time the observed likelihood ratio statistic seems not consistent with H (approximate p -value .039). In substantive terms, the observed association between age and cannabis consumption cannot be explained by an independent generative model and slight faking good manipulations for the adult group.

Example 3 (continued)

In this section we provide a full exploratory SGR analysis for the data presented in table 3. In particular, we show how it is possible to vary the parameters (γ_1^-, δ_1^-) of the fake negative distribution and evaluate how these changes effect the results of the approximate p -value. Figure 6 shows the contour plot of the approximate p -value as a function of different levels for the shape parameters γ_1^- and δ_1^- in the group of adults. More specifically, the value of parameter γ_1^- was varied at 21 distinct levels from 0.5 to 5.5 (by a 0.25 step). The same set of values was also applied for the second shape parameter δ_1^- . In this application we also changed the overall probability of faking good π^- by replacing the original value 0.1 (used in the previous example) with the new value 0.2. By contrast, all the other parameters' values were left unchanged in the SGR simulation by keeping the same values reported in the previous analysis. The results show how the value of the observed statistic, $\Lambda = 5.22$, is consistent with an independent true model ($\Lambda = 0$) that has been corrupted by a moderate amount of faking good perturbation (20%), and which is also characterized by an extreme faking pattern in the replacement distribution. This is evident from a quick inspection of figure 6 where the parameters assignments that resulted consistent with the earlier faking hypothesis are restricted to the left portion of the main diagonal ($\gamma_1^- < \delta_1^-$) in the graphical representation. By contrast, the parameters assignments corresponding to the right portion of the main diagonal ($\gamma_1^- > \delta_1^-$) are not consistent with the hypothesis. Note that these latter values represent slight faking configurations in the replacement distribution. In sum, the results are in line with a moderate faking good process which is characterized by a general property of extremeness in the way the original true values are replaced with the fake ones in the replacement distribution. That is to say, in general the chance to replace an original true value with another lower value seem to increase as a function of the distance between two values.

In what follows we present a short code example that the reader may easily manipulate to set the desired values for the parameters in the simulation study (shape parameters, overall probabilities of faking, number of runs in the SGR simulations). Note that in this exploratory setting the overall time required to complete the SGR simulation may widely vary according to the complexity (e.g., number of different values for the parameters) of the simulation design.

```
> data(smokers)
> Z <- na.omit(smokers[,c("age", "druguse")])
>
> fit0 <- polr(ordered(druguse)~1, data=Z)
> fit1 <- polr(ordered(druguse)~age, data=Z)
> lrt.obs <- -2*(logLik(fit0)-logLik(fit1)) # observed LRT
>
> ### SGR algorithm
```

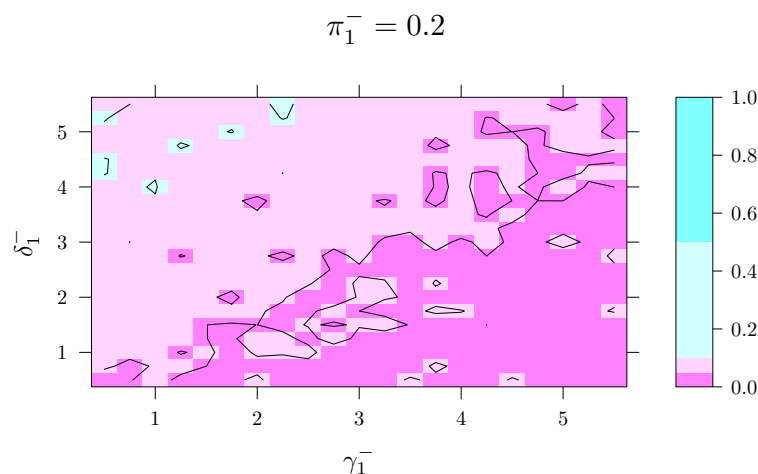


Figure 6: Contour plot for the approximate p -value as a function of shape parameter γ_1^- and shape parameter δ_1^- . In this example the overall probability of faking good for the adult group was set equal to $\pi_1^- = 0.2$. Note that the points represented to the left of the main diagonal correspond to extreme faking conditions, whereas the points represented to the right of the main diagonal correspond to slight faking conditions. The total number of runs in the SGR simulation analysis was equal to 500.

```

> PI <- .2; B <- 10 # for real simulations set B at least 500
> lrt.mc <- ga.mc <- de.mc <- p.mc <- NULL
> PM <- matrix(0,nrow(Z),ncol(Z)) # partition matrix
> PM[Z$age==1,2] <- 1
>
> for (GA in seq(.5,5.5,.5)) {
+   for (DE in seq(.5,5.5,.5)) {
+
+     Pparm <- list(p=matrix(c(0,PI),1),gam=matrix(c(0,GA),1),del=matrix(c(0,DE),1))
+
+     for (b in 1:B) {
+       Z$simdrug <- rdatagen(nrow(Z),R=matrix(1),Q=4,
+                             probs=list(c(27,5,18,10)/60))$data
+       Dx <- Z[,-2]
+       Fx <- partition.replacement(Dx,PM,Pparm=Pparm)
+
+       Tmc <- table(Fx$age,Fx$simdrug)
+       Ymc <- data.frame(list(age=gl(2,ncol(Tmc)),response=gl(ncol(Tmc),1,
+                                                                ordered=TRUE,labels=colnames(Tmc)),counts=c(Tmc[1,],Tmc[2,])))
+
+       fit0 <- polr(response~1,data=Ymc,weight=counts)
+       fit1 <- polr(response~age,data=Ymc,weight=counts)
+       statval <- -2*(logLik(fit0)-logLik(fit1))
+       lrt.mc <- c(lrt.mc,statval)
+
+       ga.mc <- c(ga.mc,GA); de.mc <- c(de.mc,DE)
+       p.mc <- c(p.mc,ifelse(statval>lrt.obs,1,0))
+     }
+   }
+ }

> LRT <- data.frame(list(gam=ga.mc,del=de.mc,lrt=lrt.mc))
> aggregate(p.mc,list(gam=LRT$gam,del=LRT$del),mean)

```

Summary, limitations, and future works

This paper illustrated the usage of a new R package, **sgr**, for simulating and analyzing ordinal fake data. As far as we know, **sgr** is the first statistical package that is devoted to the analysis of fake data. Overall, the essential characteristic of this approach is its explicit use of mathematical models and appropriate probability distributions for quantifying uncertainty in inferences based on possible fake data. Moreover, it involves the derivation of new statistical results as well as the evaluation of the implications of such new results: Are the substantive conclusions reasonable? How sensitive are the results to the modeling assumptions about the process of faking? In sum, SGR takes an interpretation perspective by incorporating in a global model all the available information about the process of faking. In this contribution we illustrated the use of **sgr** on three simple scenarios of faking. More complex examples of SGR applications can be found in [Lombardi and Pastore \(2012\)](#) and [Pastore and Lombardi \(2014\)](#).

As with many Monte Carlo-type approaches, also SGR involves simplifying assumptions that may result in lower external validity. For example, one relevant limitation regards the assumption that restricts the conditional replacement distribution to satisfy the CI property. Unfortunately, this restriction clearly limits the range of empirical faking processes that can be mimicked by the current SGR simulation procedure. In particular, because the replacement distribution under the CI assumption acts as a perturbation process for the original data, the resulting new fake data sets will in general show covariance patterns that are (on average) weaker than the ones observed for the original uncorrupted data. In general, this may not be a serious problem as different studies have shown that self-report measures under faking motivating conditions tend to have smaller variances and lower reliability (covariance) estimates than those observed for measures collected under uncorrupted conditions ([Ellingson et al., 2001](#); [Eysenck et al., 1974](#); [Hesketh et al., 2004](#); [Topping and O’Gorman, 1997](#)). However, opposite results have also been observed where simple fake good instructions tend to increase the intercorrelations between the manipulated or faked items ([Ellingson et al., 1999](#); [Galic et al., 2012](#); [Pauls and Crost, 2005](#); [Zickar and Robie, 1999](#); [Ziegler and Buehner, 2009](#)). Therefore, although encouraging, the promise of this approach should be examined across more varied conditions. We acknowledge that more work still needs to be done. We are in the process of extending **sgr** to

include new replacement distributions other than the ones presented in this article which will allow to modulate different levels of correlational patterns in the simulated fake data.

Bibliography

- A. Agresti. *Categorical Data Analysis*. Wiley, New York, NY, 1990. [p172]
- R. Beaber, A. Marston, J. Michelli, and M. Mills. A brief test for measuring malingering in schizophrenic individuals. *American Journal of Psychiatry*, 142(12):1478–1481, 1985. [p164]
- J. Cohen. *Statistical power analysis for the behavioral sciences*. Lawrence Erlbaum Associates, Hillsdale, NJ, 2nd edition, 1988. [p164]
- V. Crawford. Lying for strategic advantage: Rational and boundedly rational misrepresentation of intentions. *The American Economic Review*, 93(1):133–149, 2003. [p164]
- J. Ellingson, P. Sackett, and L. Hough. Social desirability corrections in personality measurement: Issues of applicant comparison and construct validity. *Journal Of Applied Psychology*, 84(2):155–166, APR 1999. ISSN 0021-9010. doi: {10.1037/0021-9010.84.2.155}. [p175]
- J. E. Ellingson, D. B. Smith, and P. R. Sackett. Investigating the influence of social desirability on personality factor structure. *Journal Of Applied Psychology*, 86:122–133, 2001. [p175]
- S. B. Eysenck, H. J. Eysenck, and L. Shaw. The modification of personality and lie scale scores by special ‘honesty’ instructions. *The British journal of social and clinical psychology*, 13:41–50, 1974. [p175]
- P. F. Foster, E. Fabrega, S. Karademir, N. H. Sankary, D. Mital, and J. W. Williams. Prediction of abstinence from ethanol in alcoholic recipients following liver transplantation. *Hepatology*, 25: 1469–1477, 1997. [p164]
- J. Fox. *polycor: Polychoric and Polyserial Correlations*, 2010. URL <http://CRAN.R-project.org/package=polycor>. R package version 0.7-8. [p166]
- Z. Galic, Z. Jerneic, and M. P. Kovacic. Do applicants fake their personality questionnaire responses and how successful are their attempts? A case of military pilot cadet selection. *International Journal Of Selection And Assessment*, 20(2):229–241, JUN 2012. ISSN 0965-075X. doi: {10.1111/j.1468-2389.2012.00595.x}. [p175]
- N. S. Gray, M. J. MacCulloch, J. Smith, M. Morris, and R. J. Snowden. Forensic psychology: Violence viewed by psychopathic murderers. *Nature*, 423(6939):497–498, 2003. [p164]
- J. Helton, J. Johnson, C. Salaberry, and C. Storlie. Survey of sampling based methods for uncertainty and sensitivity analysis. *Reliability Engineering and System Safety*, 91(10):1175–1209, 2006. [p164]
- B. Hesketh, B. Griffin, and D. Grayson. Applicants faking good: evidence of item bias in the NEO PI-R. *Personality And Individual Differences*, 36:1545–1558, 2004. [p175]
- C. J. Hopwood, C. A. Talbert, L. C. Morey, and R. Rogers. Testing the incremental utility of the negative impression-positive impression differential in detecting simulated personality assessment inventory profiles. *Journal of Clinical Psychology*, 64(3):338–343, 2006. [p164]
- K. Jöreskog and D. Sörbom. *PRELIS 2: User’s Reference Guide*. Scientific Software International, Inc., Lincolnwood, IL, 1996. [p165, 166, 170]
- E. L. Lehmann. The Fisher, Neyman-Pearson theories of testing hypotheses: One theory or two? *Journal of the American Statistical Association*, 88(424):1242–1249, 1993. [p169]
- L. Lombardi and M. Pastore. Sensitivity of fit indices to fake perturbation of ordinal data: A sample by replacement approach. *Multivariate Behavioral Research*, 47:519–546, 2012. [p164, 165, 175]
- E. Marshall. How prevalent is fraud? That’s a million-dollar question. *Science*, 290(5497):1662–1663, 2000. [p164]
- I. Moustaki and M. Knott. Generalized latent trait models. *Psychometrika*, 65(3):391–411, SEP 2000. ISSN 0033-3123. doi: {10.1007/BF02296153}. [p165]
- B. Muthén. A general structural equation model with dichotomous, ordered categorical and continuous latent variables indicators. *Psychometrika*, 49:115–132, 1984. [p165]

- M. Pastore and L. Lombardi. The impact of faking on Cronbach's alpha for dichotomous and ordered rating scores. *Quality & Quantity*, 48:1191–1211, 2014. doi: 10.1007/s11135-013-9829-1. [p165, 166, 167, 168, 171, 175]
- M. Pastore, L. Lombardi, and F. Mereu. Effects of malingering in self-report measures: A scenario analysis approach. In C. H. Skiadas, editor, *Recent Advances in Stochastic Modeling and Data Analysis*, pages 483–491. World Scientific Publishing, 2007. [p170]
- D. L. Paulhus. Two-component models of socially desirable responding. *Journal of Personality and Social Psychology*, 46(3):598–609, 1984. [p171]
- C. Pauls and N. Crost. Effects of different instructional sets on the construct validity of the NEO-PI-R. *Personality And Individual Differences*, 39(2):297–308, JUL 2005. ISSN 0191-8869. doi: {10.1016/j.paid.2005.01.003}. [p175]
- F. Samejima. *Estimation of Latent Ability Using a Response Pattern of Graded Scores*. Psychometric Monograph No. 17. Richmond, VA: Psychometric Society, 1969. [p165]
- G. D. Topping and J. O'Gorman. Effects of faking set on validity of the NEO-FFI. *Personality and Individual Differences*, 23(1):117–124, 1997. ISSN 0191-8869. doi: [http://dx.doi.org/10.1016/S0191-8869\(97\)00006-8](http://dx.doi.org/10.1016/S0191-8869(97)00006-8). URL <http://www.sciencedirect.com/science/article/pii/S0191886997000068>. [p175]
- S. Van der Geest and S. Sarkodie. The fake patient: A research experiment in a Ghanaian hospital. *Social Science & Medicine*, 47(9):107–120, 1998. [p164]
- 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. [p172]
- M. J. Zickar and C. Robie. Modeling faking good on personality items: An item-level analysis. *Journal of Applied Psychology*, 84:551–563, 1999. [p175]
- M. Ziegler and M. Buehner. Modeling socially desirable responding and its effects. *Educational And Psychological Measurement*, 69(4):548–565, AUG 2009. ISSN 0013-1644. doi: {10.1177/0013164408324469}. [p175]

Luigi Lombardi

Department of Psychology and Cognitive Science, University of Trento

Corso Bettini, 31, 38068 Rovereto, TN

Italy

luigi.lombardi@unitn.it

Massimiliano Pastore

Department of Developmental and Social Psychology, University of Padova

Via Venezia, 8, 35131 Padua

Italy

massimiliano.pastore@unipd.it

Web Technologies Task View

by Patrick Mair and Scott Chamberlain

Abstract This article presents the CRAN Task View on Web Technologies. We describe the most important aspects of Web Technologies and Web Scraping and list some of the packages that are currently available on CRAN. Finally, we plot the network of Web Technology related package dependencies.

Introduction

In modern Statistics and Data Science, Web technologies have become an essential tool for accessing, collecting, and organizing data from the Web. Well known companies, including Google, Amazon, Wikipedia, and Facebook, and increasingly all data providers, are providing APIs (Application Programming Interfaces) that include specifications for variables, data structures, and object classes which facilitate the interaction with other software components such as R.

Data collection from the Web can be performed at several levels: From low-level HTML/XML/JSON parsing at the one end up to calling simple functions implemented by numerous convenience packages (which, e.g., make use of API specifications internally) on the other end. A comprehensive description of Web technology strategies can be found in [Nolan and Temple Lang \(2014\)](#).

Within the R community, projects like *omegahat* (<http://www.omegahat.org/>) or rOpenSci (<http://ropensci.org/> where people can contribute through <https://github.com/ropensci/webservices>) made an effort to provide infrastructures that allow R to interact with the Web. Recently, a vast amount of corresponding Web technology packages have been developed which are now collected, organized, and structured in a *Web Technology CRAN Task View* available at <http://cran.r-project.org/web/views/WebTechnologies.html>. By setting up this task view, which we present in this paper and which will be updated regularly, we hope that this facilitates useRs to get an overview of what is available and, at the same time, it motivates future use and development of R packages that interact with the Web.

Note that not all packages listed in the Task View are on CRAN. Some of them are in the *omegahat* repository, and others on GitHub <https://github.com/> or R-Forge <https://r-forge.r-project.org/>. The corresponding package links can be found in the links section of the Task View.

Low level Web scraping

Web scraping refers to the process of extracting information from websites. In low-level scraping tasks, the two basic concepts are *retrieving* and *parsing*. The core packages in R that allow for either retrieving or parsing (or both) are **XML** (both), **RCurl** (retrieving), and **rjson/RJSONIO/jsonlite** (parsing). The **XML** package contains functions for parsing XML and HTML, and supports XPath for searching XML. A helpful function in **XML** to extract data from one or more HTML tables is `readHTMLTable()`. The **RCurl** package makes use of the `libcurl` library for transferring data using various protocols and provides a low level curl wrapper that allows one to compose general HTTP requests and provides convenient functions to fetch URLs, get/post forms, etc. and process the results returned by the Web server. This provides a great deal of control over the HTTP/FTP connection and the form of the request while providing a higher-level interface than is available just using R socket connections. It also provides tools for Web authentication. Within this context, the **httr** package (a higher level wrapper around **RCurl**) provides easy-to-use functions. JSON stands for JavaScript Object Notation and is a data interchange format becoming the most common data format on the web. The **rjson**, **RJSONIO**, and **jsonlite** packages convert R objects into JSON objects and vice-versa.

Authentication

Using Web resources can require authentication, either via API keys, OAuth, username:password combination, or via other means. Additionally, sometimes Web resources require authentication to be in the header of an http call, which requires a little bit of extra work. API keys and username:password can be combined within a URL for a call to a web resource (api key: `http://api.foo.org/?key=yourkey`; user/pass: `http://username:password@api.foo.org`), or can be specified via commands in **RCurl** or **httr**. OAuth is the most complicated authentication process, and can be most easily done using **httr** which includes demos for OAuth 1.0 (LinkedIn, Twitter, Vimeo) and demos for OAuth 2.0 (Facebook, GitHub, Google). The **ROAuth** package provides a separate R interface to OAuth.

Javascript

Javascript has become a dominant programming language for building web applications, including for response web site elements, as well as maps. An increasing number of R packages are bringing the power of Javascript to useRs. In our Task View we highlight many of these. One of these packages, not on CRAN yet, is **rCharts**, which wraps a suite of Javascript libraries for visualization. That is, **rCharts** lets you pass in typical R objects like data frames and lists and you get out Javascript visualizations in your browser (or in the “Viewer” in recent versions of RStudio’s IDE). A framework RStudio has built, called **shiny**, combines Javascript with CSS, and HTML to make building web applications with R relatively painless.

Data repositories

Providing datasets in (Open Access) repositories in order to foster reproducibility and subsequent analyses on these data is a crucial component in modern science. The importance of reproducible research in various fields has been widely discussed in over the years ([Gentleman and Temple Lang, 2004](#); [Koenker and Zeileis, 2009](#); [Peng, 2009](#); [Pebesma et al., 2012](#)). General Open Access repositories like the Dataverse network (<http://thedata.org/>) or PubMed related databases in biomedical literature (<http://www.ncbi.nlm.nih.gov/pubmed>) provide a highly important contributions to this field.

Recently, many R packages have been developed that interface Open Access data repositories using corresponding API specifications internally. In fact, this type of packages represents a large portion of all packages included in the Task View. Here, we just give a few examples from different areas. Within the area of Ecology and Evolutionary Biology we have packages such as **rgbif** and **rmbn** for biodiversity data and **rfishbase** and **rfisheries** for fishery data. In the area of Genomics **rsnps** and **rentrez** allow for the interaction with various SNP (Single-Nucleotide Polymorphism) repositories and NCBI (National Center for Biotechnology Information), respectively. In Earth Science, for example, implementations for downloading data from the Climate Reference Network (**crn** package) and obtaining data from National Centers for Environmental Prediction (**RNCEP** package).

In the Economics and Business area the **WDI** scrapes World Bank’s world development indicators, in the Finance area packages like **TFX** connects to TrueFX for free streaming real-time and historical tick-by-tick market data for interbank foreign exchange rates, and in the Marketing area the **anametrix** package is bidirectional connector to the Anametrix API.

The **rpubchem** package interfaces the PubChem collection in the Chemistry area whereas **cimis** is a package for retrieving data from CIMIS, the California Irrigation Management Information System in the area of Agriculture. To quote an example from Sports, the **nhlscrapr** compiles the NHL (National Hockey League) real time scoring system database. More packages are listed in the Task View.

Collecting data from the Web

A core aspect of Web scraping is to harvest unstructured data, often texts, from the Web. The Internet provides massive amounts of text data from sources like blogs, online newspapers, social network platforms, etc. Especially in the areas like Social Sciences and Linguistics, this type of data provides a valuable resource for research. Once the data are harvested, the **tm** package offers many functionalities to handle text data in R.

Companies such as Google, Facebook, Twitter, or Amazon provide APIs which allow analysts to retrieve data. Several convenience packages have been implemented in R which facilitate the use of these API. With respect to Google we have **RGoogleStorage** which provides programmatic access to the Google Storage API, **RGoogleDocs** interfaces the Google Documents API, **translate** provides bindings for the Google Translate API, **scholar** provides functions to extract citation data from Google Scholar, **RGoogleTrends** gives programmatic access to Google Trends data, and **RgoogleMap** allows to import Google maps into R. In order to interact with Facebook, **Rfacebook** package provides an interface to the Facebook API whereas **twitterR** and **streamR** interact with Twitter. Regarding Amazon services, the **AWS.tools** package provides access to Amazon Web Services (EC2/S3) and **MTurkR** gives access to the Amazon Mechanical Turk Requester API.

In terms of newspapers, the **GuardianR** package provides an interface to the Open Platform’s Content API of the Guardian Media Group whereas **RNYTimes** interfaces to several of the New York Times Web services for searching articles, meta-data, user-generated content, and best seller lists. Again, the packages we list here represent only a small portion of packages contained in the Task View.

Package Dependencies

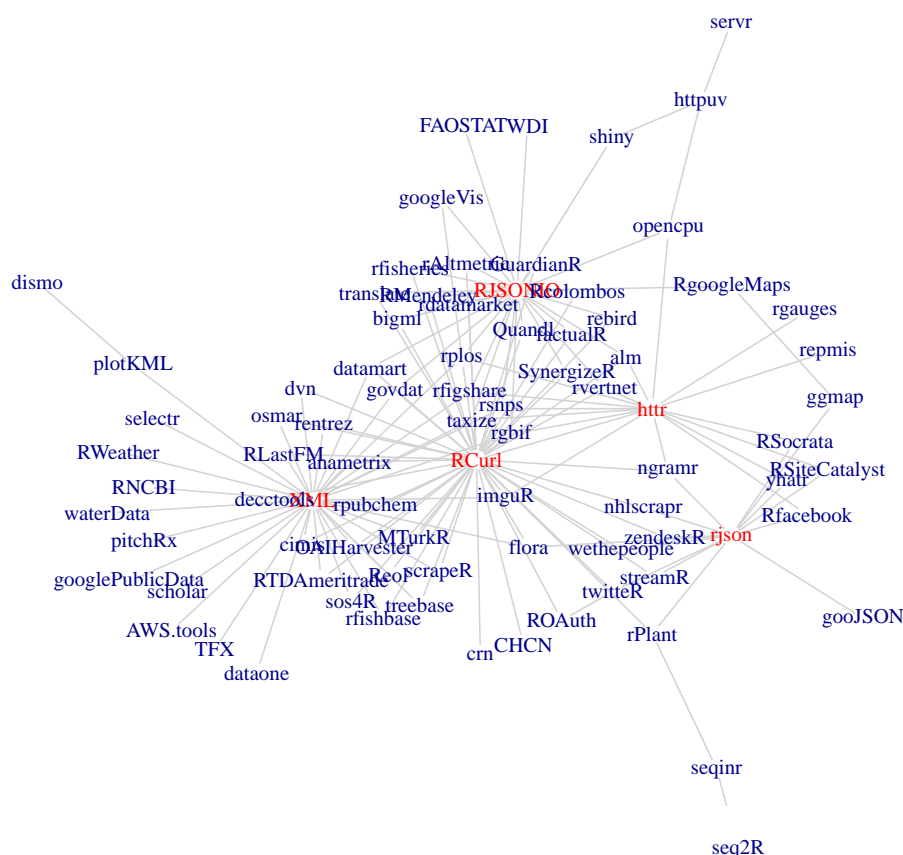


Figure 1: A network plot that shows the package dependencies and imports (only packages listed in the Web Technologies Task View are considered).

Conclusion

In this article we presented the Web Technologies Task View which is now online on CRAN. In order to give a final package overview, the network plot in Figure 1 represents all CRAN packages from the Task View including the dependency structure. The network plot, produced using **igraph**, is based on a scraping job where we harvested all corresponding package dependencies (and imports) from CRAN. Therefore, this plot does not include Task View packages that are only hosted on GitHub and Omegahat. The edges in Figure 1 reflect dependencies between packages that are listed in the Task View. We see that the low level packages such as **XML**, **RCurl**, **rjson**, **RJSONIO**, and **httr** are the core packages whose functionalities are used by many high level scraping packages. The Web Technologies Task View will be updated on a regular basis, and therefore, the network plot will change accordingly. The R code for producing the plot can be found at <https://gist.github.com/sckott/7831696>.

Bibliography

- R. Gentleman and D. Temple Lang. Statistical analyses and reproducible research. Technical Report 2, Bioconductor Project Working Papers, 2004. URL <http://biostats.bepress.com/bioconductor/paper2/>. [p179]
- R. Koenker and A. Zeileis. On reproducible econometric research. *Journal of Applied Econometrics*, 24(5): 833–847, 2009. [p179]

- D. Nolan and D. Temple Lang. *XML and Web Technologies for Data Sciences with R*. Springer, New York, 2014. [p178]
- E. Pebesma, D. D. Nüst, and R. Bivand. The R software environment in reproducible geoscientific research. *Eos Transactions American Geophysical Union*, 93(16):163–163, 2012. [p179]
- R. D. Peng. Reproducible research and Biostatistics. *Biostatistics*, 10(3):405–408, 2009. [p179]

Patrick Mair
Department of Psychology
Harvard University
USA mair@fas.harvard.edu

Scott Chamberlain
Biology Department
Simon Fraser University
Canada scott@ropensci.org

Addendum to “Statistical Software from a Blind Person’s Perspective”

by A. Jonathan R. Godfrey and Robert Erhardt

Abstract This short note explains a solution to a problem for blind users when using the R terminal under Windows Vista or Windows 7, as identified in [Godfrey \(2013\)](#). We note the way the solution was discovered and subsequent confirmatory experiments.

As part of his preparations for teaching a blind student in a statistics course, the second author read [Godfrey \(2013\)](#). He also attempted to use the R terminal in conjunction with the screen reading software his student would be using. More important though was his use of the keyboard gestures that a blind person uses in place of mouse clicks; the solution was found quite by accident by not completing the Alt+Tab key combination for switching applications.

[Godfrey \(2013\)](#) stated:

“...use of R in the terminal window of Windows Vista and Windows 7 has proved problematic for the screen reading software which can ‘lose focus’. This means the cursor is no longer able to be controlled by the blind user and the session must be curtailed. This problem is not specific to a particular screen reader, and can be replicated in other terminal windows such as the command prompt of these operating systems. At the time of writing, this problem remains and the author can only recommend using base R in batch mode to compensate.”

The loss of focus problem still exists, but the blind user can return to having a normal functioning terminal window by a single press of the Alt key alone. The first author has ensured that this solution works for a range of screen reader software, and for both Windows Vista and Windows 7. The R session does not need to be curtailed, meaning that in terms of the options for running an R job, blind users of R under Windows are back on par with their sighted colleagues.

Acknowledgement

The first author wishes to thank the editor for accepting this addendum. It is crucial that blind students and their lecturers are kept up to date with any developments that affect the way blind people can use R. This update may seem trivial to some readers, but is hugely important to blind users of R.

Bibliography

A. J. R. Godfrey. Statistical software from a blind person’s perspective: R is the best, but we can make it better. *The R Journal*, 5(1):73–80, 2013. URL <http://journal.R-project.org/archive/2013-1/godfrey.pdf>. [p182]

A. Jonathan R. Godfrey
Massey University
Palmerston North
New Zealand
a.j.godfrey@massey.ac.nz

Robert Erhardt
Wake Forest University
Winston-Salem
North Carolina
United States
erhardrj@wfu.edu

R Foundation News

by Kurt Hornik

Donations and new members

Donations

Grupo Usuarios R — Madrid, Spain

Korean R Users Group, Korea

Quan Development LLC, USA

Ravinderpal Vaid, USA

SBW Consulting, USA

New benefactors

HMS Analytical Software, Germany

New supporting institutions

SriSatish Ambati, USA

Syncfusion Inc, USA

New supporting members

Ana Maria Dobre, Romania

Hans-Michael Kaltenbach, Germany

Joris Muller, France

Jong-Hwa Shin, Korea

Kurt Hornik

WU Wirtschaftsuniversität Wien, Austria

Kurt.Hornik@R-project.org

News from the Bioconductor Project

by the Bioconductor Team

The Bioconductor project provides tools for the analysis and comprehension of high-throughput genomic data. The 824 software packages available in Bioconductor can be viewed at <http://bioconductor.org/packages/release/>. Navigate packages using 'biocViews' terms and title search. Each package has an html page with a description, links to vignettes, reference manuals, and usage statistics. Start using Bioconductor and R version 3.1.0 with

```
source("http://bioconductor.org/biocLite.R")
biocLite()
```

Install additional packages and dependencies, e.g., **deepSNV**, with

```
source("http://bioconductor.org/biocLite.R")
biocLite("deepSNV")
```

Upgrade installed packages with

```
source("http://bioconductor.org/biocLite.R")
biocLite()
```

Bioconductor 2.14 Release Highlights

Bioconductor 2.14 was released on 14 April 2014. It is compatible with R 3.1.0 and consists of 824 software packages, 200 experiment data packages, and more than 860 current annotation packages. The release includes 77 new software packages and many updates and improvements to existing packages. The [release announcement](#) includes descriptions of new packages and updated NEWS files provided by current package maintainers.

New packages continue to represent a wide variety of research areas. **Rariant** identifies single nucleotide variants (SNVs) based on the difference of binomially distributed mismatch rates between matched samples. **SomaticSignatures** identifies the mutational signatures of SNVs. Filtering of SNVs based on inheritance models, amino acid change consequence, and minor allele frequency is offered through **VariantFiltering**. The **monocle** package, currently in the 'devel' branch, preforms differential expression and time series analysis for single-cell expression experiments. CRISPR/Cas is a compelling new molecular biology technique used for gene editing; **CRISPRseek** helps find potential guide RNAs for input target sequences. **CCREPE** assesses the significance of similarity measures in 'compositional' data sets, as found in microbial abundance studies. **MIMOSA** models count data using Dirichlet-multinomial and beta-binomial mixtures with applications to single-cell assays. Machine learning methods such as SVM, Random Forest and CART are applied to RNASeq data in **MLSeq**; clustering and classification methods are used to summarize active paths in genome-scale metabolic and reaction networks in **NetPathMiner**. Bioconductor hosts a number of packages relevant to chemical compound discovery. The latest additions integrate bioinformatics and chemoinformatics into a molecular informatics platform in **Rcpi**, and performs alternating least squares analysis on chemical data in **alsace**.

In addition to these contributed packages, 'Ranges' infrastructure packages such as **GenomicRanges**, **GenomicAlignments**, and **GenomicFeatures** provide an extensive, mature and extensible framework for interacting with high throughput sequence data. One recent addition is the **GenomeInfoDb** package, which contains functions that allow translation between different chromosome sequence naming conventions (e.g., UCSC versus NCBI). Many packages rely on the Ranges infrastructure for interoperable, re-usable analysis; [Lawrence et al. \(2013\)](#) provides an introduction.

Our large collection of microarray, transcriptome and organism-specific *annotation packages* have 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. The [AnnotationHub](#), with 10,780 entries, complements our traditional offerings with diverse whole genome annotations from Ensembl, ENCODE, dbSNP, UCSC, and elsewhere.

Other activities

The Bioconductor [Git-SVN Bridge](#) allows developers to synchronize a GitHub repository with the canonical Bioconductor SVN package repository. Commits made in SVN are propagated to GitHub and vice versa. This was driven by developer requests for access to social coding features, such as issue tracking and pull requests. The service has been well received, with 73 bridges established as of June 2014.

The Bioconductor Amazon Machine Instance is now compatible with the StarCluster toolkit. This enhancement makes it straightforward to configure a cluster with nodes that communicate via MPI, SSH or Sun Grid Engine, and to control jobs via the [BiocParallel](#) and [BatchJobs](#) packages. Details are available at the AMI page http://www.bioconductor.org/help/bioconductor-cloud-ami/#using_cluster.

New Bioconductor package contributors are encouraged to consult the [Package Guidelines](#) and [Package Submission](#) sections of the Bioconductor web site, and use the new [BiocCheck](#) package, in addition to R CMD check, for guidance on conforming to Bioconductor package standards.

The Bioconductor web site advertises [training and community events](#); [mailing lists](#) 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.

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 Comput Biol*, 9(8):e1003118, 08 2013. doi: 10.1371/journal.pcbi.1003118. [p184]

The Bioconductor Team
Program in Computational Biology
Fred Hutchinson Cancer Research Center

Changes on CRAN

2013-12-01 to 2014-06-30

by Kurt Hornik and Achim Zeileis

New packages in CRAN task views

Bayesian: [BAYSTAR](#), [PReMiuM](#), [eigenmodel](#), [rstiefel](#), [spikeslab](#).

ChemPhys: [clustvarsel](#), [investr](#), [nlreg](#), [resemble](#).

ClinicalTrials: [AGSDest](#), [binomSamSize](#), [metasens](#).

Cluster: [PReMiuM](#), [clusterfly](#), [clustvarsel](#), [dendextend](#), [mixer](#), [optpart](#), [seriation](#).

Econometrics: [SemiParSampleSel](#), [mfx](#), [mnlogit](#).

Environmetrics: [Distance](#), [EnvStats](#), [RMark](#), [bReeze](#), [dsm](#), [dynatopmodel](#), [marked](#), [mrds](#).

Finance: [ChainLadder](#), [GEVStableGarch](#), [MarkowitzR](#), [Rbitcoin](#), [TAQMNGR](#), [bizdays](#), [egcm](#), [lgarch](#), [pbo](#), [tvm](#), [ycinterextra](#).

Genetics: [rmetasim](#).

HighPerformanceComputing: [Rhpc](#), [ffbase](#), [toaster](#).

MachineLearning: [RoughSets](#), [penalizedLDA](#).

MetaAnalysis: [Gmisc](#), [Metatron](#), [RcmdrPlugin.EZR](#), [extfunnel](#), [metap](#), [metasens](#), [mmeta](#), [pcnetmeta](#), [robumeta](#).

Multivariate: [clustvarsel](#), [denpro](#).

NaturalLanguageProcessing: [boilerpipeR](#), [languageR](#), [tm.plugin.alceste](#), [tm.plugin.europresse](#), [tm.plugin.lexisnexus](#), [tm.plugin.webmining](#).

NumericalMathematics: [QZ](#), [SparseM](#), [conicfit](#), [geigen](#), [ktsolve](#), [rARPACK](#).

OfficialStatistics: [MicSim](#), [foreign](#), [prevR](#), [questionr](#), [seasonal](#), [sms](#).

Optimization: [DEoptimR](#), [GA](#), [GrassmannOptim](#), [NMOF](#).

Phylogenetics: [TreePar](#).

Psychometrics: [DFIT](#), [DIFlasso](#), [TestScorer](#), [pcIRT](#), [pcaPA](#).

ReproducibleResearch: [bibtex](#), [papeR](#), [sparktex](#), [stargazer](#), [suRtex](#), [table1xls](#), [texreg](#), [tikzDevice](#), [tth](#).

Robust: [GSE](#), [OutlierDC](#), [OutlierDM](#), [RECSO](#), [RobPer](#), [RobRSVD](#), [drgee](#), [georob](#), [mblm](#), [robcor](#), [robumeta](#), [rsig](#), [ssmrob](#).

SocialSciences: [lmeSplines](#).

Spatial: [AMOEBA](#), [geospacom](#), [latticeDensity](#), [leafletR](#), [marmap](#), [seg](#), [siplab](#), [spaMM](#), [spacom](#).

SpatioTemporal: [rmatio](#).

Survival: [AdapEnetClass](#), [BGPhazard](#), [CoxRidge](#), [LogrankA](#), [MAMSE](#), [MRsurv](#), [NADA](#), [NPMLEcmprsk](#), [OutlierDC](#), [ROct](#), [SemiCompRisks](#), [SemiMarkov](#), [SurvRegCensCov](#), [YPmodel](#), [coarseDataTools](#), [kmc](#), [kmconfband](#), [psbcGroup](#), [rsig](#), [simMSM](#), [simPH](#), [smoothHR](#), [spatsurv](#), [surv2sampleComp](#), [survMisc](#), [survsim](#), [wtcrsk](#).

TimeSeries: [BAYSTAR](#), [BootPR](#), [GEVStableGarch](#), [KFKSDS](#), [LPtimeSeries](#), [MAPA](#), [MSwM](#), [MTS](#), [Sim.DiffProc](#), [TED](#), [TSdist](#), [VAR.etc](#), [bsts](#), [costat](#), [depmix](#), [depmixS4](#), [dlnm](#), [earlywarnings](#), [fractal](#), [glarma](#), [lgarch](#), [locits](#), [lomb](#), [mvcwt](#), [npst](#), [pdfetch](#), [rmaf](#), [seasonal](#), [spectral.methods](#), [stsm](#), [stsm.class](#), [tsintermittent](#), [tsoutliers](#).

WebTechnologies: [GhcnDaily](#), [RForcecom](#), [RISmed](#), [Rbitcoin](#), [RefManageR](#), [Thinknum](#), [XML2R](#), [acs](#), [ecoengine](#), [fbRanks](#), [hoarder](#), [httpRequest](#), [jsonlite](#), [okmesonet](#), [pdfetch](#), [plusser](#), [primerTree](#), [psidR](#), [pubmed.mineR](#), [raincpc](#), [rapport](#), [rbhl](#), [rbison](#), [rnoaa](#), [soilDB](#), [spocc](#), [sweSCB](#), [tm.plugin.webmining](#), [tseries](#), [weatherData](#).

(* = core package)

New contributed packages

AMOEBA: A Multidirectional Optimum Ecotope-Based Algorithm. Author: Guillermo Valles. In view: [Spatial](#).

AR1seg: Segmentation of an autoregressive Gaussian process of order 1. Authors: S. Chakar, E. Lebarbier, C. Levy-Leduc, S. Robin.

ARPobservation: Tools for simulating different methods of observing behavior based on alternating renewal processes. Author: James E. Pustejovsky.

ATmet: Advanced Tools for Metrology. Authors: S. Demeyer, A. Allard.

AmpliconDuo: Statistical Analysis Of Amplicon Data Of The Same Sample To Identify Artefacts. Authors: Anja Lange [aut, cre], Daniel Hoffmann [aut].

AntWeb: programmatic interface to the AntWeb. Author: Karthik Ram [aut, cre].

Anthropometry: Statistical methods for anthropometric data oriented towards the ergonomic design of products. Authors: Guillermo Vinue, Irene Epifanio, Amelia Simo, M. Victoria Ibanez, Juan Domingo, Guillermo Ayala.

BAMMtools: Analysis and visualization of macroevolutionary dynamics on phylogenetic trees. Authors: Dan Rabosky, Mike Grundler, Pascal Title, Carlos Anderson, Jeff Shi, Joseph Brown, Huateng Huang.

BANOVA: Hierarchical Bayesian ANOVA Models. Authors: Chen Dong, Michel Wedel.

BBRecapture: Bayesian Behavioural Capture-Recapture Models. Authors: Luca Tardella and Danilo Alunni Fegatelli.

BEQI2: Benthic Ecosystem Quality Index 2. Authors: Willem van Loon [aut, cph], Dennis Walvoort [aut, cre].

BIPOD: Bayesian Inference for Partially Observed diffusions. Author: Anders Chr. Jensen.

BMhyd: PCM for Hybridization. Author: Dwueng-Chwuan Jhwueng.

BalancedSampling: Balanced and spatially balanced sampling. Author: Anton Grafström.

BayesComm: Bayesian community ecology analysis. Author: Nick Golding.

BayesGESM: Bayesian Analysis of Generalized Elliptical Semiparametric Models. Authors: Luz Marina Rondon and Heleno Bolfarine.

BayesMed: Default Bayesian hypothesis tests for correlation, partial correlation, and mediation. Authors: Michele B. Nuijten, Ruud Wetzels, Dora Matzke, Conor V. Dolan, and Eric-Jan Wagenmakers.

BayesPen: Bayesian Penalized Credible Regions. Authors: Ander Wilson, Howard D. Bondell, and Brian J. Reich.

BerlinData: Easy access to Berlin related data. Authors: Kate McCurdy, Dirk Schumacher.

BiSEp: Toolkit to identify synthetic lethality and cell line resistance relationships based on bimodality in gene expression data. Author: Mark Wappett.

Blaunet: Calculate and Analyze Blau statuses for measuring social distance. Authors: Michael Genkin [aut], George Berry [aut], Liyuan Chen [aut], Matthew Brashears [aut].

Boom: Bayesian Object Oriented Modeling. Authors: Steven L. Scott is the sole author and creator of the BOOM project. Some code in the BOOM libraries has been modified from other open source projects. These include Cephes (obtained from Netlib, written by Stephen L. Moshier), NEWUOA (M.J.D Powell, obtained from Powell's web site), and a modified version of the R math libraries (R core development team).

BoomSpikeSlab: MCMC for spike and slab regression. Author: Steven L. Scott.

CAvariants: Correspondence Analysis Variants. Authors: Rosaria Lombardo and Eric J Beh.

CCMnet: Simulate Congruence Class Model for Networks. Authors: Ravi Goyal, with contributions from Mark S. Handcock, David R. Hunter, Carter T. Butts, Steven M. Goodreau, Pavel N. Krivitsky, Martina Morris, and Nicole Bohme Carnegie.

CCpop: One and two locus GWAS of binary phenotype with case-control-population design. Author: Shachar Kaufman.

CHAT: Clonal Heterogeneity Analysis Tool. Author: Bo Li.

CIDnetworks: Generative models for complex networks with conditionally independent dyadic structure. Authors: Beau Dabbs, Brian Junker, Mauricio Sadinle, Tracy Sweet, A.C. Thomas.

CMF: Collective matrix factorization. Authors: Arto Klami and Lauri Väre.

CMPControl: Control Charts for Conway-Maxwell-Poisson Distribution. Author: Kimberly Sellers and Luis Costa.

CORM: The Clustering of Regression Models Method. Authors: Li-Xuan Qin [aut], Jiejun Shi [cre].

CRAC: Cosmology R Analysis Code. Author: Jiayi Liu.

CRF: Conditional Random Fields. Author: Ling-Yun Wu [aut, cre].

CVD: Color Vision Deficiencies. Authors: José Gama [aut, cre, trl], Maria Nyberg [aut], Göran Högnäs [ths], Brian Foutch [ctb], Mark Grundland [ctb], Neil Dodgson [ctb].

CVcalibration: Estimation of the Calibration Equation with Error-in Observations. Authors: He Qi and Lu Tian.

ChargeTransport: Charge Transfer Rates and Charge Carrier Mobilities. Authors: Julien Idé and Guido Raos.

CircE: Circumplex models Estimation. Author: Michele Grassi.

ClickClust: Model-Based Clustering of Categorical Sequences. Author: Volodymyr Melnykov [aut, cre].

CoImp: Copula based imputation method. Authors: Francesca Marta Lilja Di Lascio, Simone Giannerini.

- CombinS**: Constructions method of rectangular PBIB and rectangular right angular PBIB(m) ($m = 4, 5$ and 7) designs. Authors: Mohamed Laib, Imane Rezgui, Zoubida Gheribi-Aoulmi and Herve Monod.
- CommunityCorrelogram**: Ecological Community Correlogram. Authors: J. Malia Andrus, Timothy Kuehlhorn, Luis F. Rodriguez, Angela D. Kent, and Julie L. Zilles.
- ConnMatTools**: Tools for working with connectivity matrices. Author: Marco Andrello.
- CorReg**: Linear regression based on linear structure between covariates. Authors: Clement Thery [aut, cre], Christophe Biernacki [ctb], Gaetan Loridant [ctb], Florian Watrin [ctb], Quentin Grimonprez [ctb], Vincent Kubicki [ctb].
- DBFTest**: DBF test of no difference between groups. Author: Christopher Minas.
- DESnowball**: Bagging with Distance-based Regression for Differential Gene Expression Analyses. Author: Yaomin Xu.
- DEoptimR**: Differential Evolution Optimization in pure R. Authors: Eduardo L. T. Conceicao and Martin Maechler [ctb]. In view: [Optimization](#).
- DFIT**: Differential Functioning of Items and Tests (DFIT) framework analyses. Author: Victor H. Cervantes. In view: [Psychometrics](#).
- DNAprofiles**: DNA profiling evidence analysis. Author: Maarten Kruijver.
- DSsim**: Distance Sampling Simulations. Author: Laura Marshall.
- Density.T.HoldOut**: Density.T.HoldOut: Non-combinatorial T-estimation Hold-Out for density estimation. Authors: Nelo Magalhães and Yves Rozenholc.
- DescTools**: Tools for descriptive statistics. Authors: Andri Signorell. Includes R source code and/or documentation previously published by (in alphabetical order): Nanina Anderegg, Tomas Aragon, Antti Arppe, Ben Bolker, Stephane Champely, Daniel Chessel, Leanne Chhay, Michael Dewey, Harold C. Doran, Stephane Dray, Charles Dupont, Jeff Enos, Claus Ekstrom, Martin Elff, John Fox, Tal Galili, Matthias Gamer, Juergen Gross, Gabor Grothendieck, Frank E. Harrell Jr, Michael Hoehle, Christian W. Hoffmann, Markus Huerzeler, Rob J. Hyndman, Pablo J. Villacorta Iglesias, David Kahle, Matthias Kohl, Mikko Korpela, Jim Lemon, Martin Maechler, Arni Magnusson, Daniel Malter, Alina Matei, David Meyer, Yongyi Min, Markus Naepflin, Derek Ogle, Sandrine Pavoine, Roland Rapold, William Revelle, Venkatraman E. Seshan, Greg Snow, Michael Smithson, Werner A. Stahel, Yves Tille, Adrian Trapletti, Kevin Ushey, Jeremy VanDerWal, Bill Venables, John Verzani, Gregory R. Warnes, Stefan Wellek, Peter Wolf, Achim Zeileis.
- Disake**: Discrete associated kernel estimators. Authors: W. E. Wansouwe, C. C. Kokonendji and D.T. Kolyang.
- DiscML**: Estimate evolutionary rates of discrete characters using maximum likelihood. Authors: Tane Kim, Weilong Hao.
- DivE**: Diversity Estimator. Authors: Daniel Laydon, Aaron Sim, Charles Bangham, Becca Asquith.
- DoubleExpSeq**: Differential Exon Usage Test for RNA-Seq data via Empirical Bayes Shrinkage of the Dispersion Parameter. Author: Sean Ruddy.
- EBEN**: Empirical Bayesian Elastic Net. Authors: Anhui Huang, Shizhong Xu, Xiaodong Cai.
- ELT**: Build Experience Life Tables. Authors: Julien Tomas, Frederic Planchet, Wassim Youssef.

- EMMIXcontrasts**: Contrasts in mixed effects for EMMIX model with random effects. Authors: Angus Ng, Geoff McLachlan, Kui Wang.
- EMMREML**: Fitting mixed models with known covariance structures. Authors: Deniz Akdemir, Okeke Uche Godfrey.
- EMP**: Expected Maximum Profit for Credit Scoring. Authors: Cristian Bravo and Thomas Verbraken.
- ERP**: Significance analysis of Event-Related Potentials data. Authors: David Causeur and Ching-Fan Sheu.
- ESGtoolkit**: Toolkit for the simulation of financial assets and interest rates models. Authors: Jean-Charles Croix, Thierry Moudiki, Frederic Planchet, Wassim Youssef.
- EWGoF**: Goodness-of-fit tests for the Exponential and two-parameter Weibull distributions. Author: Meryam Krit.
- EasyMARK**: Utility functions for working with mark-recapture data. Author: John Waller.
- EasyStrata**: Evaluation of stratified genome-wide association meta-analysis results. Author: Thomas Winkler.
- Ecfun**: Functions for **Ecdat**. Author: Spencer Graves.
- EloRating**: Animal Dominance Hierarchies by Elo Rating. Authors: Christof Neumann and Lars Kulik.
- ElstonStewart**: Elston-Stewart Algorithm. Author: Herve Perdry.
- EntropyEstimation**: Tools for the estimation of entropy and related quantities. Authors: Lijuan Cao and Michael Grabchak.
- EnviroStat**: Statistical analysis of environmental space-time processes. Authors: Nhu Le, Jim Zidek, Rick White, and Davor Cubranic, with Fortran code for Sampson-Guttorp estimation authored by Paul D. Sampson, Peter Guttorp, Wendy Meiring, and Catherine Hurley, and Runge-Kutta-Fehlberg method implementation by H.A. Watts and L.F. Shampine.
- EpiDynamics**: Dynamic Models in Epidemiology. Authors: Oswaldo Santos [aut, cre], Fernando Silveira Marques [aut].
- Evapotranspiration**: Authors: Danlu Guo, Seth Westra.
- EvoRAG**: Evolutionary Rates Across Gradients. Author: Jason T. Weir.
- ExtDist**: Extended Probability Distribution Functions. Authors: Haizhen Wu, A. Jonathan R. Godfrey, Kondaswamy Govindaraju.
- ExtremeBounds**: Extreme Bounds Analysis (EBA). Author: Marek Hlavac.
- FDRreg**: False discovery rate regression. Authors: James G. Scott, with contributions from Rob Kass and Jesse Windle.
- FDboost**: Boosting functional regression models. Author: Sarah Brockhaus.
- FGSG**: Feature grouping and selection over an undirected graph. Authors: Xiaotong Shen, Yiwen Sun, Julie Langou.
- FLIM**: Farewell's Linear Increments Model. Authors: Rune Hoff with contributions from Jon Michael Gran and Daniel Farewell.
- FLR**: Fuzzy Logic Rule Classifier. Authors: Constantinos Mavridis and Ioannis N. Athanasiadis.
- FLSSS**: Fixed Size Subset Sum Solution. Author: Charlie Wusuo Liu.

- FPDC**: PD-clustering and factor PD-clustering. Authors: Cristina Tortora and Paul D. McNicholas.
- FastHCS**: Compute the FastHCS outlyingness index. Author: Kaveh Vakili [aut, cre].
- FisHiCal**: Iterative FISH-based Calibration of Hi-C Data. Authors: Yoli Shavit, Fiona Kathryn Hamey and Pietro Lio'.
- FunChisq**: Nonparametric functional chi-square tests. Authors: Yang Zhang, Joe Song.
- FusedLasso**: Solves the generalized Fused Lasso. Author: Holger Hoefling.
- GDAtools**: A toolbox for the analysis of categorical data in social sciences, and especially Geometric Data Analysis. Author: Nicolas Robette.
- GEVStableGarch**: ARMA-GARCH/APARCH models with GEV and stable distributions. Authors: Thiago do Rego Sousa, Cira Etheowalda Guevara Otiniano and Silvia Regina Costa Lopes. In views: *Finance*, *TimeSeries*.
- GHQp**: Gauss Hermite Quadrature with pruning. Author: Freddy Hernandez Barajas.
- GLSME**: Generalized Least Squares with Measurement Error. Author: Krzysztof Bartoszek.
- GMCM**: Fast estimation of Gaussian Mixture Copula Models. Authors: Anders Ellern Bilgrau, Martin Boegsted, Poul Svante Eriksen.
- GNE**: Computation of generalized Nash equilibria. Author: Christophe Dutang.
- GPFDA**: Apply Gaussian Process in Functional data analysis. Authors: Jian Qing Shi, Yafeng Cheng.
- GPLTR**: Generalized Partially Linear Tree-based Regression Model. Authors: Cyprien Mbogning and Wilson Toussile.
- GSAgm**: Gene Set Analysis using the Gamma Method. Authors: Rama Raghavan, Alice Wang.
- GWsignif**: Genome-wide significance for whole genome sequencing studies. Authors: Changjiang Xu and Celia M.T. Greenwood.
- Gammareg**: Classic gamma regression: joint modeling of mean and shape parameters. Authors: Martha Corrales and Edilberto Cepeda-Cuervo, with the collaboration of Maria Fernanda Zarate, Ricardo Duplat and Campo Elias Pardo.
- GenABEL.data**: Contains data which is used by **GenABEL** example and test functions. Authors: Maksim Struchalin and GenABEL project developers.
- GenBinomApps**: Clopper-Pearson Confidence Interval and Generalized Binomial Distribution. Authors: Horst Lewitschnig, David Lenzi.
- GeneFeST**: Bayesian calculation of gene-specific FST from genomic SNP data. Author: Bastian Pfeifer.
- GeoGenetix**: Quantification of the effect of geographic versus environmental isolation on genetic differentiation. Authors: Filippo Botta, Gilles Guillot.
- Gmisc**: A few handy misc functions for plots, tables, and more. Author: Max Gordon. In view: *MetaAnalysis*.
- GraphPCA**: Graphical tools of histogram PCA. Authors: Brahim Brahim and Sun Makosso-Kallyth.
- HBSTM**: Hierarchical Bayesian Space-Time models for Gaussian space-time data. Authors: Pilar Munyoz, Alberto Lopez Moreno.

- HIVLifeTables**: HIV calibrated model life tables for countries with generalized HIV epidemics. Author: David J Sharrow.
- HK80**: Conversion Tools for HK80 Geographical Coordinate System. Author: Jinlong Zhang.
- HLSM**: Hierarchical Latent Space network Model. Authors: Samrachana Adhikari, Brian Junker, Tracy Sweet, Andrew C. Thomas.
- HMMCont**: Hidden Markov Model for Continuous Observations Processes. Author: Mikhail A. Beketov.
- HMMpa**: Analysing accelerometer data using hidden Markov models. Authors: Vitali Witowski, Ronja Foraita.
- HSAUR3**: A Handbook of Statistical Analyses Using R (3rd Edition). Authors: Brian S. Everitt and Torsten Hothorn.
- HUM**: Compute HUM value and visualize ROC curves. Authors: Natalia Novoselova, Junxi Wang, Jialiang Li, Frank Pessler, Frank Klawonn.
- Hankel**: Univariate non-parametric two-sample test based on empirical Hankel transforms. Author: Daniel Kolbe.
- HiClimR**: Hierarchical Climate Regionalization: An Improved Hierarchical Clustering in R for Climate Regionalization. Authors: Hamada S. Badr [aut, cre], Benjamin F. Zaitchik [aut], Amin K. Dezfouli [aut].
- HiCseg**: Detection of domains in HiC data. Author: Celine Levy-Leduc.
- HiLMM**: Estimation of heritability in high dimensional Linear Mixed Models. Author: Anna Bonnet.
- ICGE**: Estimation of number of clusters and identification of atypical units. Authors: Itziar Irigoien [aut, cre], Concepcion Arenas [aut].
- ICsurv**: Semiparametric regression analysis of interval-censored data. Authors: Christopher S. McMahan and Lianming Wang.
- InventorymodelPackage**: Author: Alejandro Saavedra Nieves.
- IsingFit**: Fitting Ising models using the eLasso method. Authors: Claudia van Borkulo, Sacha Epskamp, with contributions from Alexander Robitzsch.
- IsingSampler**: Sampling methods and distribution functions for the Ising model. Author: Sacha Epskamp.
- IsoCI**: Confidence intervals for current status data based on transformations and bootstrap. Authors: Byeong Yeob Choi, Jason P. Fine and M. Alan Brookhart.
- JBTools**: JB's tools and helper functions. Author: Jannis v. Buttlar.
- KANT**: Identify and sort genes overexpressed. Author: Noemie Robil.
- KATforDCEMRI**: Kinetic analysis and visualization of DCE-MRI data. Authors: Gregory Z. Ferl, Georges Hankov.
- KFKSDS**: Kalman Filter, Smoother and Disturbance Smoother. Author: Javier López-de-Lacalle. In view: *TimeSeries*.
- LINselect**: Selection of linear estimators. Authors: Yannick Baraud, Christophe Giraud, Sylvie Huet.
- LOGICOIL**: Multi-state prediction of coiled-coil oligomeric state. Authors: Thomas L. Vincent, Peter J. Green and Derek N. Woolfson.

- LPS:** Linear Predictor Score, for binary inference from multiple continuous variables. Author: Sylvain Mareschal.
- LPStimeSeries:** Learned Pattern Similarity and Representation for Time Series. Authors: Learned Pattern Similarity (LPS) for time series by Mustafa Gokce Baydogan, Ensemble of regression trees by Andy Liaw and Matthew Wiener. In view: *TimeSeries*.
- LS2Wstat:** A Multiscale Test of Spatial Stationarity for LS2W processes. Authors: Sarah Taylor [aut], Matt Nunes [aut, cre], Idris Eckley [ctb, ths].
- LSAfun:** Applied Latent Semantic Analysis (LSA) functions. Author: Fritz Guenther [aut, cre].
- LogisticDx:** Diagnostic tests for logistic regression models. Author: Chris Dardis.
- MAPA:** Multiple Aggregation Prediction Algorithm. Authors: Nikolaos Kourentzes and Fotios Petropoulos. In view: *TimeSeries*.
- MC2toPath:** Translates information from netcdf files with MC2 output into inter-PVT transitions. Authors: Dave Conklin and Emilie Henderson.
- MCMC.OTU:** Bayesian analysis of multivariate counts data. Author: Mikhail V. Matz.
- MIICD:** Multiple Imputation for interval censored data regression. Author: Marc Delord.
- MILC:** Microsimulation Lung Cancer (MILC) model. Author: Stavroula A. Chrysanthopoulou.
- MMMS:** Multi-Marker Molecular Signature for Treatment-specific Subgroup Identification. Authors: Lin Li, Tobias Guennel, Scott Marshall, Leo Wang-Kit Cheung.
- MTS:** All-purpose toolkit for analyzing multivariate time series (MTS) and estimating multivariate volatility models. Author: Ruey S. Tsay. In view: *TimeSeries*.
- MXM:** Discovering multiple, statistically-equivalent signatures. Authors: Ioannis Tsamardinos, Vincenzo Lagani, Giorgos Athineou.
- ManyTests:** Multiple testing procedures of Cox (2011) and Wang and Cox (2007). Author: Christiana Kartsonaki.
- MarkowitzR:** Statistical significance of the Markowitz portfolio. Author: Steven E. Pav [aut, cre]. In view: *Finance*.
- Metatron:** Meta-analysis for Classification Data and Correction to Imperfect Reference. Author: Huiling Huang. In view: *MetaAnalysis*.
- Methplot:** Visualize the methylation patterns. Author: Xin Yang.
- MiST:** Mixed effects Score Test for continuous outcomes. Authors: Jianping Sun, Yingye Zheng, and Li Hsu.
- MicSim:** Performing continuous-time microsimulation. Author: Sabine Zinn. In view: *OfficialStatistics*.
- Mposterior:** Robust and Scalable Bayes via a Median of Subset Posterior Measures. Author: Sanvesh Srivastava.
- MultiCNVDetect:** Multiple Copy Number Variation Detection. Authors: Hao Lin, Qiang Kou.
- NHMMfdr:** Compute FDR under dependence using NHMM. Author: Pei Fen Kuan.
- NLPutils:** Natural Language Processing Utilities. Author: Kurt Hornik [aut, cre].

- NPS:** Convenience functions and tests for working with the Net Promoter Score (NPS). Author: Brendan Rocks.
- Nemenyi:** Implementation of the Nemenyi post-hoc test to find the groups of data that differ after a statistical test of multiple comparisons. Author: Christian Guckelsberger.
- NetSim:** A Social Networks Simulation Tool in R. Author: Christoph Stadtfeld.
- NormPsy:** Normalisation of psychometric tests. Authors: Cecile Proust-Lima, Viviane Philipps.
- NormalLaplace:** The Normal Laplace Distribution. Authors: David Scott, Jason Shicong Fu and Simon Potter.
- ORCI:** Several confidence intervals for the odds ratio. Author: Libo Sun.
- OceanView:** Visualisation of Oceanographic Data and Model Output. Author: Karline Soetaert.
- OrdNor:** Concurrent generation of ordinal and normal data with given correlation matrix and marginal distributions. Authors: Anup Amatya and Hakan Demirtas.
- OutbreakTools:** Basic tools for the analysis of disease outbreaks. Authors: The Hack-out team (In alphabetic order: David Aanensen, Marc Baguelin, Paul Birrell, Simon Cauchemez, Anton Camacho, Caroline Colijn, Anne Cori, Xavier Didelot, Ken Eames, Christophe Fraser, Simon Frost, Niel Hens, Joseph Hugues, Thibaut Jombart, Lulla Opatowski, Oliver Ratmann, Samuel Soubeyrand, Marc Suchard, Jacco Wallinga, Rolf Ypma).
- OutlierDM:** Outlier detection for replicated high-throughput data. Authors: Soo-Heang Eo [aut, cre], HyungJun Cho [aut]. In view: [Robust](#).
- PBC:** Product of Bivariate Copulas (PBC). Authors: Van Trung Pham and Gildas Mazo, with contributions from R Core team, Nash, Zhu, Byrd, Lu-Chen and Nosedal.
- PBD:** Protracted birth-death model of diversification. Author: Rampal S. Etienne.
- PCAmixdata:** Multivariate analysis for a mixture of quantitative and qualitative data. Authors: Marie Chavent and Vanessa Kuentz and Amaury Labenne and Benoit Liquet and Jerome Saracco.
- PCDSpline:** Semiparametric regression analysis of panel count data using monotone splines. Authors: Bin Yao and Lianming Wang.
- PCGSE:** Principal Component Gene Set Enrichment. Author: H. Robert Frost.
- PCPS:** Principal Coordinates of Phylogenetic Structure. Author: Vanderlei Julio Debastiani.
- PEMM:** A Penalized EM algorithm incorporating missing-data mechanism. Authors: Lin Chen and Pei Wang.
- PGM2:** Recursive method for construction of nested resolvable designs and uniform designs associated. Authors: Mohamed Laib, Abba Boudraa and Zoubida Gheribi-Aoulmi.
- PHeval:** Evaluation of the proportional hazards assumption with a standardized score process. Author: Cecile Chauvel.
- PIGE:** Self contained gene set analysis for gene- and pathway-environment interaction analysis. Authors: Benoit Liquet, Therese Truong.
- PIGShift:** Polygenic Inverse Gamma rate Shifts. Author: Joshua G. Schraiber.
- PMCMR:** Calculate Pairwise Multiple Comparisons of Mean Rank Sums. Author: Thorsten Pohlert.

- PSMix**: Population structure inference using mixture model. Authors: Baolin Wu [aut, cph], Nianjun Liu [aut, cph], Hongyu Zhao [aut, cph], Jose Gama [cre].
- PTE**: Personalized Treatment Evaluator. Authors: Adam Kapelner, Justin Bleich.
- Paneldata**: Linear models for panel data. Author: Zaghdoudi Taha.
- ParetoPosStable**: Computing, fitting and validating the Pareto Positive Stable distribution. Authors: Antonio Jose Saez-Castillo [aut, cre], Faustino Prieto [aut], Jose Maria Sarabia [aut].
- PedCNV**: An implementation for association analysis with CNV data. Authors: Meiling Liu, Sungho Won and Weicheng Zhu.
- Peptides**: Calculate indices and theoretical physicochemical properties of peptides and protein sequences. Authors: Daniel Osorio, Paola Rondon-Villarreal and Rodrigo Torres.
- PerMallows**: Permutations and Mallows distributions. Author: Ekhine Irurozki.
- PermAlgo**: Permutational algorithm to simulate survival data. Authors: Marie-Pierre Sylvestre, Thad Edens, Todd MacKenzie, Michal Abrahamowicz. In view: *Survival*.
- PharmPow**: Pharmacometric Power calculations for mixed study designs. Authors: Frank Kloprogge and Joel Tarning.
- PoisNor**: Simultaneous generation of multivariate data with Poisson and normal marginals. Authors: Anup Amatya and Hakan Demirtas.
- PopED**: Population (and individual) optimal Experimental Design. Authors: Andrew C. Hooker [aut, cre, trl, cph], Sebastian Ueckert [aut] (MATLAB version), Marco Foracchia [aut] (O-Matrix version), Joakim Nyberg [aut] (MATLAB version), Eric Stroemberg [ctb] (MATLAB version).
- PortRisk**: Portfolio Risk Analysis. Authors: Tamal Kanti Panja, Sourish Das, Rajeswaran Viswanathan.
- Power2Stage**: Power and Sample size distribution of 2-stage BE studies via simulations. Authors: Detlew Labes [aut, cre], Helmut Schuetz [ctb].
- QuantifQuantile**: Estimation of conditional quantiles using optimal quantization. Authors: Isabelle Charlier and Davy Paindaveine and Jerome Saracco.
- R4CDISC**: Read CDISC data files. Author: Ippei Akiya.
- R4CouchDB**: An R convenience layer for CouchDB. Author: Thomas Bock.
- RADami**: Phylogenetic Analysis of RADseq Data. Author: Andrew L. Hipp.
- RAHRS**: Data fusion filters for Attitude Heading Reference System (AHRS) with several variants of the Kalman filter and the Mahoney and Madgwick filters. Authors: Jose' Gama [aut, cre, trl], Vlad Maximov [aut], Sebastian O.H. Madgwick [aut], Alain Barraud [ctb], Göran Högnäs [ths].
- RAPIDR**: Reliable Accurate Prenatal non-Invasive Diagnosis R package. Author: Kitty Lo.
- RApiSerialize**: R API Serialization. Authors: Dirk Eddelbuettel, Junji Nakano, Ei-ji Nakama, and R Core (original code).
- REdaS**: Companion Package to the Book "R: Einführung durch angewandte Statistik". Author: Marco Johannes Maier [cre, aut].
- RGENERATE**: A tool for generation random variable time series using the tools of 'vars' or 'RMAWGEN'. Author: Emanuele Cordano.

- RI2by2**: Randomization inference for treatment effects on a binary outcome. Author: Joseph Rigdon.
- RImageJROI**: Read ImageJ Region of Interest (ROI) files. Authors: David C Sterratt [aut, cph, cre], Mikko Vihtakari [aut, cph].
- RJSONLD**: Semantic packaging tools for standard analytics. Author: Joseph Dureau.
- ROC632**: Construction of diagnostic or prognostic scoring system and internal validation of its discriminative capacities based on ROC curve and 0.633+ bootstrap resampling. Author: Y. Foucher.
- ROct**: Time-dependent ROC curve estimation and adaptation to the relative survival context. Authors: Y. Foucher, K. Trebern-Launay and M. Lorent. In view: *Survival*.
- RPublica**: ProPublica API Client. Author: Thomas J. Leeper.
- RPushbullet**: R interface to the wonderful Pushbullet service. Author: Dirk Eddelbuettel.
- RSNPset**: Authors: Chanhee Yi, Alexander Sibley, and Kouros Owzar.
- RSelenium**: R bindings for Selenium WebDriver. Author: John Harrison.
- RSpincalc**: Converting between attitude representations: DCM, Euler angles, Quaternions, and Euler vectors. Authors: Jose' Gama [aut, cre], John Fuller [aut, cph], Paolo Leva [aut, cph], Göran Högnäs [ths].
- RTextureMetrics**: Functions for calculation of texture metrics for Grey Level Co-occurrence Matrices. Author: Hans-Joachim Klemmt.
- RWBP**: Detects spatial outliers using a Random Walk on Bipartite Graph. Authors: Sigal Shaked and Ben Nasi.
- RWebLogo**: plotting custom sequence logos. Author: Omar Wagih.
- RYoudaoTranslate**: Functions to translate English words into Chinese. Author: Ke-Hao Wu.
- RankResponse**: Ranking Responses in a Single Response Question or a Multiple Response Question. Authors: Hsiuying Wang, Yu-Jun Lin.
- RapidPolygonLookup**: Polygon lookup using kd trees. Authors: Markus Loecher and Madhav Kumar.
- Rchoice**: Discrete Choice (Binary, Poisson and Ordered) Models with Random Parameters. Author: Mauricio Sarrias.
- RcmdrPlugin.NMBU**: R Commander Plug-In for statistics at NMBU. Authors: Kristian Hovde Liland, Solve Sæbø.
- Rcpp11**: R and C++11. Authors: Romain Francois [aut, cre], Kevin Ushey [aut], John Chambers [ctb].
- RcppRedis**: Rcpp bindings for Redis using the hiredis library. Author: Dirk Eddelbuettel.
- RefFreeEWAS**: EWAS using reference-Free DNA methylation mixture deconvolution. Authors: E. Andres Houseman.
- RefManageR**: Straightforward Bib_TEX and Bib_LAT_EX Bibliography Management. Author: Mathew W. McLean [aut, cre]. In view: *WebTechnologies*.
- RegClust**: Cluster analysis via regression coefficients. Authors: Weichao Bao, Xin Tong, Meredith Ray, Hongmei Zhang.
- RelValAnalysis**: Relative Value Analysis. Author: Ting-Kam Leonard Wong [aut, cre].

- Reot:** Empirical Orthogonal Teleconnections in R. Authors: Tim Appelhans, Florian Detsch, Thomas Nauss.
- ReporteRs:** Microsoft Word, Microsoft Powerpoint and HTML documents generation from R. Author: David Gohel.
- ReporteRsjars:** External jars required for package **ReporteRs**. Author: David Gohel.
- Rhpc:** High-Performance Computing. Authors: Junji Nakano and Ei-ji Nakama. In view: *HighPerformanceComputing*.
- RobRSVD:** Robust Regularized Singular Value Decomposition. Authors: Lingsong Zhang and Chao Pan. In view: *Robust*.
- Rothermel:** Rothermel fire spread model for R. Authors: Giorgio Vacchiano, Davide Ascoli.
- RoughSets:** Data Analysis Using Rough Set and Fuzzy Rough Set Theories. Authors: Lala Septem Riza, Andrzej Janusz, Chris Cornelis, Francisco Herrera, Dominik Slezak, and Jose Manuel Benitez. In view: *MachineLearning*.
- Rpdb:** Read, write, visualize and manipulate PDB files. Author: Julien Idé.
- Rphylip:** An R interface for PHYLIP. Authors: Liam J. Revell, Scott A. Chamberlain.
- Rrdrand:** Generate Physical Random Numbers on Intel CPUs with the RdRand instruction. Authors: Eiji Nakama, Junji Nakano.
- RsimMosaic:** R Simple IMage Mosaic creation library. Author: Alberto Krone-Martins.
- Rsomoclu:** R package for somoclu. Authors: Peter Wittek [aut], Shichao Gao [cre].
- Rtsne:** T-distributed Stochastic Neighbor Embedding using Barnes-Hut implementation. Author: Jesse Krijthe.
- Rttf2pt1:** Package for ttf2pt1 program. Authors: Winston Chang, Andrew Weeks, Frank M. Siegert, Mark Heath, Thomas Henlick, Sergey Babkin, Turgut Uyar, Rihardas Hepas, Szalay Tamas, Johan Vromans, Petr Titera, Lei Wang, Chen Xiangyang, Zvezdan Petkovic, Rigel, I. Lee Hetherington.
- Ruchardet:** Detect character encoding. Author: Heewon Jeon.
- Rvcg:** Manipulations of triangular meshes (smoothing, quadric edge collapse decimation, im- and export of various mesh file-formats, cleaning, etc.) based on the VCGLIB API. Author: Stefan Schlager; the authors of VCGLIB for the included version of the code.
- SCBmeanfd:** Simultaneous Confidence Bands for the Mean of Functional Data. Author: David Degras.
- SCI:** Standardized Climate Indices such as SPI, SRI or SPEI. Authors: Lukas Gudmundsson and James H. Stagge.
- SCORER2:** An algorithm for distinguishing parallel dimeric and trimeric coiled-coil sequences. Authors: Craig T. Armstrong, Thomas L. Vincent, Peter J. Green and Derek N. Woolfson.
- SNFtool:** Similarity Network Fusion. Authors: Bo Wang, Aziz Mezlini, Feyyaz Demir, Marc Fiume, Zhuowen Tu, Michael Brudno, Benjamin Haibe-Kains, Anna Goldenberg.
- SPAr:** Perform rare variants association analysis based on summation of partition approaches. Author: Ruixue Fan.
- STEPCAM:** ABC-SMC inference of the STEPCAM model. Authors: Thijs Janzen and Fons van der Plas.
- STPGA:** Selection of Training Populations by Genetic Algorithm. Author: Deniz Akdemir.

- Sample.Size:** Sample size calculation. Authors: Wei Jiang, Jonathan Mahnken, Matthew Mayo.
- SemiCompRisks:** Parametric and semi-parametric analyses of semi-competing risks data. Authors: Kyu Ha Lee and Sebastien Haneuse. In view: *Survival*.
- SeqFeatR:** Calculates possible epitopes and co-mutations. Author: Bettina Budeus.
- ShrinkCovMat:** Shrinkage Covariance Matrix Estimators. Author: Anestis Touloumis.
- SimRAD:** Simulations to predict the number of loci expected in RAD and GBS approaches. Authors: Olivier Lepais [aut, cre], Jason Weir [aut].
- SimSeq:** Nonparametric Simulation of RNA-Seq Data. Author: Samuel Benidt.
- SocialMediaMineR:** A Social Media Search and Analytic Tool. Author: Marco Toledo Bastos.
- SocialNetworks:** Generates social networks based on distance. Authors: Glenna Nightingale, Peter Nightingale.
- SoftClustering:** Soft Clustering Algorithms. Author: G. Peters.
- SpecsVerification:** Forecast verification routines for the SPECS FP7 project. Author: Stefan Siegert [aut, cre].
- StableEstim:** Estimate the 4 parameters of stable law using different methods. Authors: Tarak Kharrat, Georgi N. Boshnakov.
- StereoMorph:** Stereo Camera Calibration and Reconstruction. Authors: Aaron Olsen, Annat Haber.
- Storm:** Write Storm Bolts in R using the Storm Multi-Language Protocol. Author: Allen Day.
- SubLasso:** Gene selection using Lasso for Microarray data with user-defined genes fixed in model. Authors: Fengfeng Zhou, Youxi Luo, Qinghan Meng, Ruiquan Ge, Guoqin Mai, Jikui Liu.
- Sunder:** Quantification of the effect of geographic versus environmental isolation on genetic differentiation. Authors: Filippo Botta, Casper Eriksen, Gilles Guillot.
- Surrogate:** Evaluation of surrogate endpoints in clinical trials. Authors: Wim Van der Elst, Ariel Alonso and Geert Molenberghs.
- SurvRegCensCov:** Weibull Regression for a Right-Censored Endpoint with Interval-Censored Covariate. Authors: Stanislas Hubeaux and Kaspar Rufibach. In view: *Survival*.
- TAQMNGR:** Manage tick-by-tick transaction data. Authors: Francesco Calvori, Fabrizio Cipollini, Giampiero M. Gallo. In view: *Finance*.
- TED:** Turbulence time series Event Detection and classification. Authors: Yanfei Kang, Danijel Belusic and Kate Smith-Miles. In view: *TimeSeries*.
- TFDEA:** Technology Forecasting using Data Envelopment Analysis functions. Author: ETA Research Group at Portland State University.
- TFMPvalue:** Efficient and accurate p -value computation for Position Weight Matrices. Author: Ge Tan.
- TSdist:** Distance Measures for Time Series data. Authors: Usue Mori, Alexander Mendiburu, J.A. Lozano. In view: *TimeSeries*.
- Table1Heatmap:** Table 1 Heatmap. Author: Philip C Schouten.

- TaoTeProgramming:** Illustrations from Tao Te Programming. Author: Pat Burns.
- TcGSA:** Time-course Gene Set Analysis. Author: Boris P. Hejblum [aut, cre].
- TeachNet:** Fits neural networks to learn about back propagation. Author: Georg Steinbuss.
- TopKLists:** Inference, aggregation and visualization for top- k ranked lists. Authors: Michael G. Schimek, Eva Budinska, Jie Ding, Karl G. Kugler, Vendula Svendova, Shili Lin.
- TrackReconstruction:** Reconstruct animal tracks from magnetometer, accelerometer, depth and optional speed data. Author: Brian Battaile.
- TreatmentSelection:** Evaluate Treatment Selection Biomarkers. Authors: Marshall Brown and Holly Janes.
- TurtleGraphics:** Turtle graphics in R. Authors: Anna Cena [aut], Marek Gagolewski [aut], Marcin Kosinski [aut], Natalia Potocka [aut], Barbara Zogala-Siudem [aut, cre].
- UBCRM:** Functions to simulate and conduct dose escalation phase I studies. Author: Benjamin Esterni with contribution from Baboukar Mane.
- VAR.etc:** VAR modelling: estimation, testing, and prediction. Author: Jae. H. Kim. In view: *TimeSeries*.
- VIMGUI:** Visualization and Imputation of Missing Values. Authors: Daniel Schopfhauser, Matthias Templ, Andreas Alfons, Alexander Kowarik, Bernd Prantner.
- VNM:** Multiple objective optimal design. Authors: Seung Won Hyun, Weng Kee Wong, and Yarong Yang.
- VdgRsm:** Variance Dispersion and Fraction of Design Space Plots. Authors: Patchanok Srisuradetchai, John J. Borkowski.
- VideoComparison:** Video comparison tool. Authors: Silvia Espinosa, Joaquin Ordieres, Antonio Bello, Jose Maria Perez.
- VoxR:** Metrics extraction of trees from T-LiDAR data. Authors: Bastien Lecigne, Sylvain Delagrangue and Christian Messier.
- Wats:** Wrap Around Time Series graphics. Authors: Will Beasley [aut, cre], Joe Rodgers [aut], Matthew Schuelke [ctb], Ronnie Coleman [ctb], Mark Joseph Lachowicz [ctb].
- WikipediR:** A MediaWiki API wrapper. Author: Oliver Keyes.
- WrightMap:** Wright Map: IRT item-person map with ConQuest integration. Authors: David Torres Irribarra and Rebecca Freund.
- XBRL:** Extraction of business financial information from XBRL documents. Authors: Roberto Bertolusso and Marek Kimmel.
- XNomial:** Exact Goodness-Of-Fit Test For Multinomial Data With Fixed Probabilities. Author: Bill Engels.
- Xmisc:** Xiaobei's miscellaneous classes and functions. Author: Xiaobei Zhao [aut, cre, cph].
- aRpsDCA:** Arps Decline Curve Analysis in R. Author: Derrick Turk [aut, cre, cph].
- accrual:** Bayesian Accrual Prediction. Authors: Yu Jiang, Steve Simon, Matthew S. Mayo, Rama Raghavan, Byron J. Gajewski.
- acm4r:** Align-and-Count Method comparisons of RFLP data. Authors: Andrea Benedetti, Sahir Rai Bhatnagar, Xiaofei Zhao.
- acss:** Algorithmic Complexity for Short Strings. Authors: Nicolas Gauvrit [aut], Henrik Singmann [aut, cre], Fernando Soler Toscano [ctb], Hector Zenil [ctb].

- acss.data**: Data Only: Algorithmic Complexity of Short Strings (Computed via Coding Theorem Method). Authors: Fernando Soler Toscano [aut], Nicolas Gauvrit [aut], Hector Zenil [aut], Henrik Singmann [aut, cre].
- additivityTests**: Additivity tests in the two way ANOVA with single sub-class numbers. Authors: Marie Simeckova [aut], Thomas Rusch [aut], Petr Simecek [cre].
- adhoc**: Calculate ad hoc distance thresholds for DNA barcoding identification. Author: Gontran Sonet.
- aemo**: Download and process AEMO price and demand data. Authors: Imanuel Costigan [aut, cre], Australian Energy Market Operator [cph].
- aidar**: Tools for reading AIDA (<http://aida.freehep.org/>) files into R. Author: Andreas Pfeiffer.
- algstat**: Algebraic statistics in R. Authors: David Kahle and Luis Garcia-Puente.
- anominate**: alpha-NOMINATE Ideal Point Estimator. Authors: Royce Carroll, Christopher Hare, Jeffrey B. Lewis, James Lo, Keith T. Poole, and Howard Rosenthal.
- apmsWAPP**: Pre- and Postprocessing for AP-MS data analysis based on spectral counts. Author: Martina Fischer.
- apsimr**: Edit, run and evaluate APSIM simulations from R easily. Author: Bryan Stanfill.
- argparser**: Command-line argument parser. Author: David JH Shih.
- arnie**: “Arnie” box office records 1982-2014. Author: Imanuel Costigan [aut, cre].
- assertthat**: Easy pre and post assertions. Author: Hadley Wickham [aut, cre].
- astrochron**: An R Package for Astrochronology. Author: Stephen Meyers.
- autoencoder**: An implementation of sparse autoencoder for automatic learning of representative features from unlabeled data. Authors: Eugene Dubossarsky (project leader, chief designer), Yuriy Tyshetskiy (design, implementation, testing).
- babel**: Ribosome profiling data analysis. Authors: Adam B. Olshen, Richard A. Olshen, Barry S. Taylor.
- bartMachine**: Bayesian Additive Regression Trees. Authors: Adam Kapelner and Justin Bleich.
- bcpmeta**: Bayesian Multiple Changepoint Detection Using Metadata. Author: Yingbo Li.
- bdvis**: Biodiversity Data Visualizations. Authors: Vijay Barve, Javier Otegui.
- beepR**: Easily Play Notification Sounds on any Platform. Author: Rasmus Bååth.
- benford.analysis**: Benford Analysis for data validation and forensic analytics. Author: Carlos Cinelli.
- berryFunctions**: function collection related to hydrology, zooming and shapefiles. Author: Berry Boessenkool.
- bezier**: Bezier Curve and Spline Toolkit. Author: Aaron Olsen.
- bigalgebra**: BLAS routines for native R matrices and big.matrix objects. Authors: Michael J. Kane, Bryan Lewis, and John W. Emerson.
- bigpca**: PCA, transpose and multicore functionality for big.matrix objects. Author: Nicholas Cooper.
- bigsplines**: Big Splines (smoothing splines for large samples). Author: Nathaniel E. Helwig.

- bilan**: Bilan water balance model. Authors: T. G. Masaryk Water Research Institute, p.r.i.: Ladislav Kasperek, Martin Hanel, Stanislav Horacek, Petr Maca, Adam Vizina.
- binda**: Multi-Class Discriminant Analysis using Binary Predictors. Authors: Sebastian Gibb and Korbinian Strimmer.
- bingat**: Binary Graph Analysis Tools. Authors: Terrence Brooks, Berkley Shands, Skye Buckner-Petty, Patricio S. La Rosa, Elena Deych, William D. Shannon.
- bioPN**: Simulation of deterministic and stochastic biochemical reaction networks using Petri Nets. Authors: Roberto Bertolusso and Marek Kimmel.
- biotools**: Tools for Biometry and Applied Statistics in Agricultural Science. Author: Anderson Rodrigo da Silva.
- biplotbootGUI**: Bootstrap on Classical Biplots. Authors: Ana Belen Nieto Librero, Purificacion Galindo Villardon.
- birdring**: Methods to analyse ring re-encounter data. Authors: Fraenzi Korner-Nievergelt, Rob Robinson.
- birk**: MA Birk functions. Author: Matthew A Birk.
- blockmatrix**: Tools to solve algebraic systems with partitioned matrices. Author: Emanuele Cordano.
- blowtorch**: Constrained optimization via stochastic gradient descent. Author: Steven Pollack.
- bmmix**: Bayesian multinomial mixture. Author: Thibaut Jombart.
- bold**: Interface to Bold Systems API. Author: Scott Chamberlain [aut, cre].
- boolean3**: Boolean Binary Response Models. Author: Jason W. Morgan.
- boostR**: A modular framework to bag or boost any estimation procedure. Author: Steven Pollack.
- bootLR**: Bootstrapped confidence intervals for (negative) likelihood ratio tests. Authors: Keith A. Marill and Ari B. Friedman.
- brainR**: Helper functions to **misc3d** and **rgl** packages for brain imaging. Author: John Muschelli III.
- breakpoint**: Multiple Break-Point Detection via the Cross-Entropy Method. Authors: Priyadarshana W.J.R.M. and Georgy Sofronov.
- bshazard**: Nonparametric Smoothing of the Hazard Function. Authors: Paola Rebora, Agus Salim, Marie Reilly.
- bsts**: Bayesian structural time series. Author: Steven L. Scott. In view: *TimeSeries*.
- bujar**: Buckley-James Regression for Survival Data with High-Dimensional Covariates. Authors: Zhu Wang and others (see COPYRIGHTS).
- cAIC4**: Conditional Akaike information criterion for lme4. Authors: Benjamin Saefken and David Ruegamer, with contributions from Sonja Greven and Thomas Kneib.
- capm**: Companion Animal Population Management. Authors: Oswaldo Santos [aut, cre], Marcos Amaku [ctb], Fernando Ferreira [ctb].
- caseMatch**: Identify similar cases for qualitative case studies. Author: Rich Nielsen.
- cati**: Community Assembly by Traits: Individuals and beyond. Authors: Adrien Taudiere, Cyrille Violle with contribution of Francois Munoz.

- ccda**: Combined Cluster and Discriminant Analysis. Authors: Solt Kovacs, Jozsef Kovacs, Peter Tanos.
- cgam**: Constrained Generalized Additive Model. Authors: Mary C. Meyer and Xiyue Liao.
- checkmate**: Fast and versatile argument checks. Authors: Michel Lang, Bernd Bischl.
- chipPCR**: Toolkit of helper functions to pre-process amplification data. Authors: Stefan Roediger, Michal Burdukiewicz.
- choroplethr**: Functions to simplify the creation of choropleths (thematic maps) in R. Authors: Ari Lamstein [cre, aut], Brian P Johnson [ctb, frontend animation code].
- chromoR**: Analysis of chromosomal interactions data (correction, segmentation and comparison). Author: Yoli Shavit.
- clere**: CLERE methodology for simultaneous variables clustering and regression. Authors: Loic Yengo, Mickael Canouil.
- clickstream**: Analyzes clickstreams based on Markov chains. Author: Michael Scholz.
- clustMD**: Model based clustering for mixed data. Author: Damien McParland.
- cna**: Coincidence Analysis (CNA). Author: Mathias Ambuehl.
- colourlovers**: R client for the COLOURlovers API. Author: Thomas J. Leeper.
- comato**: Analysis of Concept Maps. Author: Andreas Muehling.
- confreq**: Configural Frequencies Analysis Using Loglinear Modeling. Authors: Joerg-Henrik Heine, R.W. Alexandrowicz and some package testing by Mark Stemmler.
- conicfit**: Algorithms for fitting circles, ellipses and conics based on the work by Prof. Nikolai Chernov. Authors: Jose' Gama [aut, cre], Nikolai Chernov [aut, cph], Göran Högnäs [ths]. In view: *NumericalMathematics*.
- cooccur**: Probabilistic Species Co-occurrence Analysis in R. Authors: Daniel M. Griffith, Joseph A. Veech, and Charles J. Marsh.
- corclass**: Correlational Class Analysis. Author: Andrei Boutyline.
- correlate**: Multivariate Data Generation by Permutation. Authors: Pascal van Kooten, Gerko Vink.
- cosmosR**: Author: Maxime Wack.
- covreg**: A simultaneous regression model for the mean and covariance. Authors: Xiaoyue Niu and Peter Hoff.
- cpca**: Methods to perform Common Principal Component Analysis (CPCA). Authors: Andrey Ziyatdinov [aut, cre], Samir Kanaan-Izquierdo [aut], Nickolay T. Trendafilov [aut], Alexandre Perera-Lluna [aut].
- crackR**: Probabilistic damage tolerance analysis for fatigue cracking of metallic aerospace structures. Author: Keith Halbert.
- crossval**: Generic Functions for Cross Validation. Author: Korbinian Strimmer.
- csn**: Closed-skew normal distribution. Author: Eugene Girtcius.
- cstar**: Substantive significance testing for regression estimates and marginal effects. Author: Justin Esarey.
- csvread**: Fast Specialized CSV File Loader. Author: Sergei Izrailev.

- cubfits**: Codon Usage Bias Fits. Authors: Wei-Chen Chen [aut, cre], Russell Zaretzki [aut], William Howell [aut], Drew Schmidt [aut], Michael Gilchrist [aut], Students REU13 [ctb].
- cutoffR**: CUTOFF: A Spatio-temporal Imputation Method. Authors: Lingbing Feng, Gen Nowak, Alan. H. Welsh, Terry. J. O'Neill.
- cuttlefish.model**: Performs LPUE standardization and stock assessment of the English Channel cuttlefish stock using a two-stage biomass model. Authors: Michael Gras and Jean-Paul Robin.
- dagbag**: Learning directed acyclic graphs (DAGs) through bootstrap aggregating. Authors: Ru Wang, Jie Peng.
- dams**: Dams in the United States from the National Inventory of Dams (NID). Author: Gopi Goteti.
- datalist**: Convert data sets for import into JAGS, WinBUGS and OpenBUGS and to generate data.frames for predicting the effects of particular variables. Author: Joe Thorley [aut, cre].
- dawai**: Discriminant analysis with additional information. Authors: David Conde, Miguel A. Fernandez, Bonifacio Salvador.
- deepnet**: Deep learning toolkit in R. Author: Xiao Rong.
- dendextend**: Extending R's dendrogram functionality. Authors: Tal Galili [aut, cre, cph], Gavin Simpson [ctb], jefferis [ctb] (imported code from his dendroextras package), Marco Gallotta [ctb], plannapus [ctb], Gregory [ctb], R core team [ctb] (Other than the Infrastructure, some code came from their examples), Kurt Hornik [ctb], Uwe Ligges [ctb], Yoav Benjamini [ths]. In view: [Cluster](#).
- dendextendRcpp**: Faster dendrogram manipulation using [Rcpp](#). Authors: Tal Galili [aut, cre, cph], Romain Francois [ctb], Dirk Eddelbuettel [ctb], Kevin Ushey [ctb], Yoav Benjamini [ths].
- dendsort**: Sorting and reordering dendrogram nodes. Author: Ryo Sakai.
- dfexplore**: Explore data.frames by plotting NA and classes of each variable. Author: Joris Muller.
- diskmemoiser**: Disk Memoisation For R. Author: Farzad Noorian.
- dissUtils**: Utilities for making pairwise comparisons of multivariate data. Author: Benjamin N. Taft.
- dnet**: Omics data integrative analysis in terms of network, evolution and ontology. Authors: Hai Fang and Julian Gough.
- docopt**: Commandline interface specification language. Author: Edwin de Jonge.
- domino**: Domino Data Lab R console bindings. Author: Jacek Glodek.
- dplyr**: A grammar of data manipulation. Authors: Hadley Wickham, Romain Francois.
- drgee**: Doubly Robust Generalized Estimating Equations. Authors: Johan Zetterqvist, Arvid Sjölander, with contributions from Alexander Ploner. In view: [Robust](#).
- drsmooth**: Dose-Response Modeling with Smoothing Splines. Authors: Greg Hixon [aut, cph], Anne Bichteler [aut, cre], Chad Thompson [ctb], Liz Abraham [ctb].
- dualScale**: Dual Scaling Analysis of Multiple Choice Data. Authors: Jose G. Clavel, Shizuiko Nishisato and Antonio Pita.

- dynatopmodel**: Implementation of the Dynamic TOPMODEL hydrological model. Authors: Peter Metcalfe, based on Fortran code by Keith Beven and Jim Freer. In view: *Environmetrics*.
- dynia**: Fit Dynamic Intervention Model. Authors: Jinkun Xiao and A.I. McLeod.
- easingr**: Fetch the Cleveland Federal Reserve Bank easing policy tool data. Author: Matt Barry.
- easynls**: Easy nonlinear model. Author: Emmanuel Arnhold.
- ebSNP**: Genotyping and SNP calling using single-sample next generation sequencing data. Author: Na You.
- ecoengine**: Programmatic interface to the API serving UC Berkeley's Natural History Data. Author: Karthik Ram [aut, cre]. In view: *WebTechnologies*.
- edeR**: Email Data Extraction Using R. Author: Jaynal Abedin.
- edmr**: Empirical differentially methylated regions calculation. Authors: Sheng Li [aut, cre, cph], Francine Garrett-Bakelman [ctb], Altuna Akalin [ctb], Paul Zumbo [ctb], Ari Melnick [ctb], Chris Mason [ctb, ths].
- eegAnalysis**: Tools for analysis and classification of electroencephalography (EEG) data. Authors: Murilo Coutinho Silva, George Freitas von Borries.
- egcm**: Engle-Granger cointegration models. Author: Matthew Clegg. In view: *Finance*.
- eiwild**: Ecological Inference with individual and aggregate data. Author: Thomas Schlesinger.
- erpR**: Event-related potentials (ERP) analysis, graphics and utility functions. Authors: Giorgio Arcara, Anna Petrova.
- etasFLP**: Estimation of an ETAS model. Mixed FLP (Forward Likelihood Predictive) and ML estimation of non-parametric and parametric components of the ETAS model for earthquake description. Authors: Marcello Chiodi [aut, cre], Giada Adelfio [aut].
- eulerian**: Find eulerian paths from graphs. Authors: Ashis Saha, with contribution from Jaewoo Kang.
- evolvability**: Calculation of evolvability parameters. Author: Geir H. Bolstad.
- expp**: Spatial analysis of extra-pair paternity. Authors: Mihai Valcu, Lotte Schlicht.
- extWeibQuant**: Estimate the lower extreme quantile with the censored Weibull MLE and censored Weibull Mixture. Author: Yang (Seagle) Liu.
- fSRM**: Social Relations Analyses with roles ("Family SRM"). Authors: Felix Schönbrodt, Lara Stas, Tom Loeys.
- factorplot**: Authors: Dave Armstrong.
- falcon**: Finding Allele-specific Copy Number in Next-Generation Sequencing Data. Authors: Hao Chen and Nancy R. Zhang.
- fastHICA**: Hierarchical Independent Component Analysis: a multi-scale sparse non-orthogonal data-driven basis. Authors: Piercesare Secchi, Simone Vantini, and Paolo Zanini.
- fastM**: Fast Computation of Multivariate M-estimators. Authors: Lutz Duembgen, Klaus Nordhausen, Heike Schuhmacher.
- fat2Lpoly**: Two-locus Family-based Association Test with Polytomic Outcome. Authors: Alexandre Bureau and Jordie Croteau.
- fcd**: Fused Community Detection. Authors: Yang Feng, Richard J. Samworth and Yi Yu.

- federalregister**: Client package for the U.S. Federal Register API. Author: Thomas J. Leeper.
- fifer**: A collection of miscellaneous functions. Author: Dustin Fife.
- findpython**: Python tools to find an acceptable python binary. Authors: Trevor L Davis and Paul Gilbert.
- finiteruinprob**: Computation of the probability of ruin within a finite time horizon. Authors: Benjamin Baumgartner [aut, cre], Riccardo Gatto [ctb, ths].
- flowfield**: Forecasts future values of a univariate time series. Author: Kyle A. Caudle.
- frair**: Functional response analysis in R. Author: Daniel Pritchard.
- freestats**: Statistical algorithms used in common data mining course. Author: Xiaoyao Yang.
- freqparcoord**: Novel Methods for Parallel Coordinates. Authors: Norm Matloff and Yingkang Xie.
- freqweights**: Working with frequency tables. Author: Emilio Torres-Manzanera.
- frm**: Regression analysis of fractional responses. Author: Joaquim J.S. Ramalho.
- fslr**: Wrapper functions for FSL (FMRIB Software Library) from Functional MRI. of the Brain (FMRIB). Author: John Muschelli.
- funreg**: Functional Regression for Irregularly Timed Data. Authors: John Dziak [aut, cre], Mariya Shiyko [aut].
- gRapfa**: Acyclic Probabilistic Finite Automata. Authors: Smitha Ankinakatte and David Edwards.
- gSeg**: Graph-Based Change-Point Detection (g-Segmentation). Authors: Hao Chen and Nancy R. Zhang.
- gamboostMSM**: Estimating multistate models using gamboost(). Author: Holger Reulen.
- gamlss.spatial**: Fit spatial data in **gamlss**. Authors: Fernanda De Bastiani, Mikis Stasinopoulos.
- gconcord**: Concord method for Graphical Model Selection. Author: Sang-Yun Oh Kshitij Khare Bala Rajaratnam.
- gdalUtils**: Wrappers for the Geospatial Data Abstraction Library (GDAL) Utilities. Authors: Jonathan Asher Greenberg and Matteo Mattiuzzi.
- genasis**: Global ENvironmental ASsessment Information System (GENASIS) computational tools. Authors: Jiri Kalina, Jana Klanova, Ladislav Dusek, Tom Harner, Jana Boruvkova, Jiri Jarkovsky.
- gendata**: Generate and modify synthetic datasets. Author: Francis Huang.
- geoCount**: Analysis and Modeling for Geostatistical Count Data. Author: Liang Jing.
- gfcanalysis**: Tools for working with Hansen et al. 2013 Global Forest Change dataset. Author: Alex Zvoleff [aut, cre].
- ggtern**: An extension to **ggplot2**, for the creation of ternary diagrams. Author: Nicholas Hamilton.
- ggvis**: Interactive grammar of graphics. Authors: RStudio, Inc.
- glarma**: Generalized Linear Autoregressive Moving Average Models. Authors: William T.M. Dunsmuir, Cenanning Li, and David J. Scott. In view: *TimeSeries*.

- glcm**: Calculate textures from grey-level co-occurrence matrices (GLCMs) in R. Author: Alex Zvoleff [aut, cre].
- glmx**: Generalized Linear Models Extended. Authors: Achim Zeileis [aut, cre], Roger Koenker [aut], Philipp Doebler [aut].
- gofit**: Tests of fit for some probability distributions. Authors: Elizabeth Gonzalez-Estrada, Jose A. Villasenor-Alva.
- gofitest**: Classical Goodness-of-Fit Tests for Univariate Distributions. Authors: Julian Faraway [aut], George Marsaglia [aut], John Marsaglia [aut], Adrian Baddeley [aut, cre].
- gramEvol**: Grammatical Evolution For R. Authors: Farzad Noorian, Anthony Mihirana de Silva.
- greport**: Graphical Reporting for Clinical Trials. Author: Frank E Harrell Jr.
- groc**: Generalized Regression on Orthogonal Components. Authors: M. Bilodeau and P. Lafaye de Micheaux.
- grppenalty**: Concave 1-norm and 2-norm group penalty in linear and logistic regression. Author: Dingfeng Jiang.
- gset**: Group Sequential Design in Equivalence Studies. Author: Fang Liu.
- gsscopu**: Copula Density and 2-D Hazard Estimation using Smoothing Splines. Author: Chong Gu.
- h2o**: H2O R Interface. Authors: Anqi Fu, Tom Kraljevic and Petr Maj, with contributions from the 0xdata team.
- hamlet**: Hierarchical Optimal Matching and Machine Learning Toolbox. Author: Teemu Daniel Laajala.
- hasseDiagram**: Drawing Hasse diagram. Author: Krzysztof Ciomek.
- hawkes**: Hawkes process simulation and calibration toolkit. Author: Riadh Zaatour.
- hazus**: Damage functions from FEMA's HAZUS software for use in modeling financial losses from natural disasters. Author: Gopi Goteti.
- hcci**: Interval estimation for the parameters of linear models with heteroskedasticity (Wild Bootstrap). Authors: Pedro Rafael Diniz Marinho [aut, cre], Francisco Cribari Neto [aut, ctb].
- hdi**: High-Dimensional Inference. Authors: Lukas Meier, Nicolai Meinshausen.
- heatmap3**: A improved heatmap package. Authors: Shilin Zhao, Yan Guo, Quanhu Sheng, Yu Shyr.
- helsinki**: Helsinki open data R tools. Authors: Juuso Parkkinen, Leo Lahti, Joona Lehtomaki.
- hflights**: Flights that departed Houston in 2011. Author: Hadley Wickham.
- hglasso**: Learning graphical models with hubs. Author: Kean Ming Tan.
- hiddenf**: The hidden F-test for nonadditivity. Authors: Jason A. Osborne and Christopher T. Franck.
- hoardeR**: Author: Daniel Fischer. In view: [WebTechnologies](#).
- hpoPlot**: Functions to plot graphviz style graphs of sets of HPO terms. Author: Daniel Greene.
- hqmisc**: Miscellaneous convenience functions and datasets. Author: Hugo Quené [aut, cre].

- hrrr**: Horizontal rule for the R language. Author: Leonardo Di Donato.
- htmltools**: Tools for HTML. Authors: RStudio, Inc.
- hybridEnsemble**: Build, deploy and evaluate a Hybrid Ensemble. Authors: Michel Ballings, Dauwe Vercamer, and Dirk Van den Poel.
- hypergea**: Hypergeometric tests. Author: Markus Boenn.
- iC10**: A copy number and expression-based classifier for breast tumours. Author: Oscar M Rueda.
- iC10TrainingData**: Training datasets for **iC10** package. Author: Oscar M Rueda.
- iDynoR**: Analysis of iDynoMiCS Simulation Results. Authors: Kieran Alden, Jan-Ulrich Kreft.
- iRepro**: Reproducibility for Interval-Censored Data. Author: Jelena Kovacic.
- iScreen**: image-based High-Throughput RNAi Screening Analysis Tool. Author: Rui Zhong.
- icapca**: Mixed ICA/PCA. Author: Roger Woods.
- icd9**: tools for working with ICD-9 codes, and finding comorbidities. Author: Jack O. Wasey [aut, cre].
- imputeR**: A General Imputation Framework in R. Authors: Lingbing Feng, Gen Nowak, Alan. H. Welsh, Terry. J. O'Neill.
- in2extRemes**: Into the **extRemes** Package. Author: Eric Gilleland.
- interventionalDBN**: Interventional Inference for Dynamic Bayesian Networks. Author: Simon Spencer.
- invGauss**: Threshold regression that fits the (randomized drift) inverse Gaussian distribution to survival data. Author: Hakon K. Gjessing.
- io**: A unified framework for input-output operations in R. Author: David JH Shih.
- ipdw**: Interpolation by Inverse Path Distance Weighting. Author: Joseph Stachelek.
- isingLenzMC**: Monte Carlo for classical Ising Model. Author: Mehmet Suzen.
- isotonic.pen**: Penalized Isotonic Regression in one and two dimensions. Authors: Mary C Meyer, Jiwen Wu, and Jean D. Opsomer.
- iterpc**: Efficient iterator for permutations and combinations. Authors: Randy Lai [aut, cre], Martin Broadhurst [aut].
- ivfixed**: Instrumental fixed effect panel data model. Author: Zaghdoudi Taha.
- ivlewbcl**: Uses heteroscedasticity to estimate mismeasured and endogenous regressor models. Author: Alan Fernihough.
- ivpack**: Instrumental Variable Estimation. Author: Dylan Small.
- jackstraw**: Non-parametric Jackstraw for Principal Component Analysis. Author: Neo Christopher Chung.
- jointPm**: Risk estimation using the joint probability method. Authors: Feifei Zheng, Michael Leonard, Seth Westra.
- jsonlite**: A smarter JSON encoder/decoder for R. Authors: Jeroen Ooms, Duncan Temple Lang, Jonathan Wallace. In view: [WebTechnologies](#).

- kimisc**: Kirill's miscellaneous functions. Author: Kirill Mueller.
- kintone**: kintone REST client package for R. Author: Ryu Yamashita.
- kmc**: Kaplan–Meier estimator with constraints for right censored data – a recursive computational algorithm. Authors: Yifan Yang, Mai Zhou. In view: *Survival*.
- knnIndep**: Independence tests and benchmarks. Author: Sebastian Dümcke.
- kyotil**: Utility Functions by Youyi & Krisz. Authors: Youyi Fong, Krisztian Sebestyen.
- lambda.tools**: Tools for modeling data with functional programming. Author: Brian Lee Yung Rowe.
- lar**: History of labour relations. Authors: Richard Zijdemman [aut, cre], Josemiguel Lana-Berasain [ctb].
- lassoscore**: high-dimensional inference with the penalized score test. Author: Arend Voorman.
- lctools**: Local Correlation, Spatial Inequalities and other tools. Author: Stamatis Kalogirou.
- leafletR**: Interactive web-maps based on the Leaflet JavaScript library. Author: Christian Graul. In view: *Spatial*.
- letsR**: Tools for data handling and analysis in macroecology. Authors: Bruno Vilela and Fabricio Villalobos.
- lgarch**: Simulation and estimation of log-GARCH models. Author: Genaro Sucarrat. In views: *Finance*, *TimeSeries*.
- linkim**: Linkage information based genotype imputation method. Authors: Yi Xu and Jixiang Wu.
- lllrcr**: Local Log-linear Models for Capture-Recapture. Author: Zach Kurtz.
- lmms**: Linear mixed effect model splines for modelling and analysis of time course data. Authors: Jasmin Straube, Kim-Anh Le Cao and Emma Huang with contributions of Dominique Gorse.
- loa**: Various plots, options and add-ins for use with **lattice**. Author: Karl Ropkins.
- localsolver**: R API to LocalSolver. Authors: Walerian Sokolowski [aut, cre, cph], Wit Jakuczun [aut, cph], Natalia Okinczyc [aut], Bogumil Kaminski [aut].
- loe**: Local Ordinal Embedding. Authors: Yoshikazu Terada, Ulrike von Luxburg.
- lpmodeler**: Modeler for linear programs (LP) and mixed integer linear programs (MILP). Author: Cyrille Szymanski [aut].
- lsgl**: Linear sparse group lasso. Author: Martin Vincent.
- ltbayes**: Simulation-Based Bayesian Inference for Latent Traits of Item Response Models. Author: Timothy R. Johnson.
- mGSZ**: Gene set analysis based on GSZ-scoring function and asymptotic p -value. Authors: Pashupati Mishra, Petri Toronen.
- maSAE**: Mandallaz' model-assisted small area estimators. Authors: Andreas Dominik Cullmann [aut, cre], Daniel Mandallaz [ctb], Alexander Francis Massey [ctb].
- magrittr**: A forward-pipe operator for R. Authors: Stefan Milton Bache and Hadley Wickham.
- mailR**: A utility to send emails from R. Author: Rahul Premraj.

- mapStats**: Geographic display of survey data statistics. Author: Samuel Ackerman.
- maptpx**: MAP estimation of topic models. Author: Matt Taddy.
- mbest**: Moment-Based Estimation for Hierarchical Models. Author: Patrick O. Perry.
- mcheatmaps**: Multiple matrices heatmap visualization. Authors: Thierry Chenard [aut], Rafael Najmanovich [aut, cre].
- mdatools**: Multivariate data analysis for chemometrics. Author: Sergey Kucheryavskiy.
- memgene**: Spatial pattern detection in genetic distance data using Moran's Eigenvector Maps. Authors: Pedro Peres-Neto, Paul Galpern.
- metaRNASeq**: Meta-analysis of RNA-seq data. Authors: Guillemette Marot, Florence Jaffrezic, Andrea Rau.
- metap**: Meta-analysis of significance values. Author: Michael Dewey. In view: *MetaAnalysis*.
- metasens**: Advanced statistical methods to model and adjust for bias in meta-analysis. Authors: Guido Schwarzer [aut, cre], James Carpenter [aut], Gerta Rücker [aut]. In views: *ClinicalTrials*, *MetaAnalysis*.
- mewAvg**: A Fixed Memory Moving Expanding Window Average. Authors: Adam L. Pintar and Zachary H. Levine.
- mfx**: Marginal Effects, Odds Ratios and Incidence Rate Ratios for GLMs. Author: Alan Fernihough. In view: *Econometrics*.
- miceadds**: Some additional multiple imputation functions, especially for **mice**. Author: Alexander Robitzsch [aut, cre].
- mime**: Map filenames to MIME types. Author: Yihui Xie.
- minque**: Linear Mixed Model Analyses. Author: Jixiang Wu.
- miscset**: Miscellaneous Tools Set. Author: Sven E. Templer.
- mixlm**: Mixed model ANOVA and statistics for education. Authors: Kristian Hovde Liland, Solve Sæbø.
- mme**: Multinomial Mixed Effects Models. Authors: E. Lopez-Vizcaino, M.J. Lombardia and D. Morales.
- mnlogit**: Multinomial Logit Model. Authors: Wang Zhiyu, Asad Hasan. In view: *Econometrics*.
- monitoR**: Acoustic template detection in R. Authors: Sasha D. Hafner and Jon Katz, with code for the Fourier transform from the **seewave** package (by Jerome Sueur, Thierry Aubin, and Caroline Simonis), and code for the readMP3 function from the **tuneR** package (by Uwe Ligges).
- morgenstemning**: Color schemes compatible with red-green color perception difficulties. Authors: Matthias Geissbuehler, James Manton.
- morse**: MOdelling tools for Reproduction and Survival data in Ecotoxicology. Authors: Marie Laure Delignette-Muller [aut, cre], Philippe Ruiz [aut, cre], Sandrine Charles [aut], Wandrille Duchemin [ctb], Christelle Lopes [ctb], Philippe Veber [ctb].
- mph**: Multiscale persistent homology. Author: Samuel Gerber.
- msos**: Datasets and Functions used in Multivariate Statistics: Old School by John Marden. Authors: John Marden [aut, cph] and James Balamuta [cre, ctb, com].

- multicon**: Multivariate Constructs. Author: Ryne A. Sherman.
- multigroup**: Methods for multigroup data analysis. Authors: Aida Eslami, El Mostafa Qannari, Stephanie Bougeard, Gaston Sanchez.
- multinbmod**: Regression analysis of overdispersed correlated count data. Author: Ivonne Solis-Trapala.
- munsellinterpol**: Interpolation of Munsell renotation data to convert any hue and chroma values to CIE xyY, CIE XYZ, sRGB, CIE Lab or CIE Luv. Authors: Jose Gama [aut, cre, trl], Paul Centore [aut, cph], Göran Högnäs [ths].
- mvctm**: Multivariate Variance Components Tests for Multilevel Data. Author: Denis Larocque.
- mvnfast**: Fast multivariate normal methods. Authors: Matteo Fasiolo, using the C++ parallel RNG of Thijs van den Berg and Ziggurat algorithm of Jens Maurer and Steven Watanabe (boost).
- mwa**: Causal inference in spatiotemporal event data. Authors: Sebastian Schutte and Karsten Donnay.
- mycobacrVR**: Integrative immunoinformatics for Mycobacterial diseases in R platform. Authors: Deepika Kulshreshtha, Rupanjali Chaudhuri, S. Ramachandran.
- nat**: NeuroAnatomy Toolbox for Analysis of 3D Image Data. Authors: Greg Jefferis and James Manton.
- nat.utils**: File System Utility Functions for NeuroAnatomy Toolbox. Author: Gregory Jefferis.
- ncdf.tools**: Easier ncdf file handling. Author: Jannis v. Buttlar.
- ncdf4.helpers**: Helper functions for use with the **ncdf4** package. Author: David Bronaugh for the Pacific Climate Impacts Consortium (PCIC).
- networkDynamicData**: Dynamic network datasets. Author: Skye Bender-deMoll [cre, aut].
- networkreporting**: Tools for using network reporting estimators. Authors: Dennis Feehan, Matthew Salganik.
- neuroim**: Reading, writing and representing brain imaging data. Author: Bradley R. Buchsbaum.
- ngram**: An n -gram Babblar. Authors: Drew Schmidt [aut, cre], Christian Heckendorf [aut].
- nlsMicrobio**: Nonlinear regression in predictive microbiology. Authors: Florent Baty and Marie-Laure Delignette-Muller.
- nmcdR**: Non-parametric Multiple Change-points Detection. Authors: Changliang Zou, Lancezhange.
- noncensus**: U.S. Census Regional and Demographic Data. Author: John A. Ramey.
- normtest**: Tests for Normality. Authors: Ilya Gavrilov, Ruslan Pusev.
- npbr**: Nonparametric boundary regression. Authors: Abdelaati Daouia, Thibault Laurent, Holsuk Noh.
- npcp**: Some nonparametric tests for change-point detection in (multivariate) observations. Author: Ivan Kojadinovic.
- nplr**: N-Parameter Logistic Regression. Authors: Frederic Commo [aut, cre], Brian M. Bot [aut].

- numOSL**: Numeric routines for optically stimulated luminescence dating. Author: Peng Jun.
- obs.agree**: Assess agreement between observers. Authors: Teresa Henriques, Luis Antunes and Cristina Costa-Santos.
- ocedata**: Oceanographic datasets for Oce. Author: Dan Kelley.
- openxlsx**: '.xlsx' reading, writing and editing. Authors: Alexander Walker [aut, cre], Luca Braglia [ctb].
- optR**: Optimization Toolbox for solving linear systems. Author: Prakash (PKS Prakash).
- optiRum**: Miscellaneous functions for finance / credit risk analysts. Author: Stephanie Locke [aut, cre].
- orca**: Counting graphlet orbits in sparse graphs. Authors: Tomaz Hocevar, Janez Demsar.
- pSI**: Specificity Index Statistic. Authors: Xiaoxiao Xu, Alan B. Wells, David OBrien, Arye Nehorai, Joseph D. Dougherty.
- paf**: Attributable Fraction Function for Censored Survival Data. Author: Li Chen.
- paleobioDB**: Downloading, visualizing and processing data from Paleobiology Database. Authors: Sara Varela [aut, cre], Javier González Hernández [aut], Luciano Fabris Sgarbi [aut].
- paleofire**: Analysis of sedimentary charcoal records from the Global Charcoal Database to reconstruct past biomass burning. Author: Global Paleofire Working Group.
- palinsol**: Insolation for palaeoclimate studies. Author: Michel Crucifix.
- panelAR**: Estimation of Linear AR(1) Panel Data Models with Cross-Sectional Heteroskedasticity and/or Correlation. Author: Konstantin Kashin.
- parallelMCMCcombine**: Methods for combining independent subset Markov chain Monte Carlo (MCMC) posterior samples to estimate a posterior density given the full data set. Authors: Alexey Miroshnikov, Erin Conlon.
- pbo**: Probability of Backtest Overfitting. Author: Matt Barry. In view: [Finance](#).
- pcIRT**: IRT models for polytomous and continuous item responses. Author: Christine Hohensinn. In view: [Psychometrics](#).
- pcg**: Preconditioned Conjugate Gradient Algorithm for solving $Ax = b$. Authors: B N Mandal and Jun Ma.
- pcnetmeta**: Methods for patient-centered network meta-analysis. Authors: Lifeng Lin, Jing Zhang, and Haitao Chu. In view: [MetaAnalysis](#).
- pdfetch**: Fetch economic and financial time series data from public sources. Author: Abiel Reinhart. In views: [TimeSeries](#), [WebTechnologies](#).
- pdmmod**: Proximal/distal modeling framework for Pavlovian conditioning phenomena. Author: Chloe Bracis.
- peacots**: Periodogram Peaks in Correlated Time Series. Author: Stilianos Louca.
- performanceEstimation**: An infra-structure for performance estimation of predictive models. Author: Luis Torgo.
- phenability**: Nonparametric Stability Analysis. Author: Leonardo Castelo Branco.

- phreeqc**: R interface to the phreeqc geochemical modeling program. Authors: S.R. Charlton, D.L. Parkhurst, and C.A.J. Appelo, with contributions from D. Gillespie for Chipmunk BASIC and S.D. Cohen, A.C. Hindmarsh, R. Serban, D. Shumaker, and A.G. Taylor for CVODE/SUNDIALS.
- phyloland**: Modelling Competitive Exclusion and Limited Dispersal in a Statistical Phylogeographic Framework. Authors: Louis Ranjard, Marie Paturel.
- phyreg**: Implements the Phylogenetic Regression of Grafen (1989). Author: Alan Grafen.
- pipeR**: Specialized, high-performance pipeline operators. Author: Kun Ren.
- pkgKitten**: Create simple packages which do not upset R CMD check. Author: Dirk Eddelbuettel.
- plot2groups**: Plot scatter points for two groups of values. Author: Fuquan Zhang [aut, cre].
- plot3D**: Plotting multi-dimensional data. Author: Karline Soetaert.
- plot3Drgl**: Plotting multi-dimensional data using **rgl**. Author: Karline Soetaert.
- plotMCMC**: MCMC Diagnostic Plots. Authors: Arni Magnusson [aut, cre], Ian Stewart [aut].
- plusser**: A Google+ Interface for R. Author: Christoph Waldhauser [aut, cre]. In view: *WebTechnologies*.
- pmcgd**: Authors: Antonio Punzo and Paul D. McNicholas.
- pollstR**: R client for the Huffpost Pollster API. Authors: Jeffrey B. Arnold [aut, cre], Thomas J. Leeper [aut].
- popRange**: A spatially and temporally explicit forward genetic simulator. Author: Kimberly F. McManus.
- pquantmalarials**: Web tool for estimating under-five deaths caused by poor-quality anti-malarials in sub-Saharan Africa. Author: J. Patrick Renschler.
- praktikum**: Kvantitatiivsete meetodite praktikumi asjad / Functions used in the course "Quantitative methods in behavioural sciences" (SHPH.00.004), University of Tartu. Author: Kenn Konstabel.
- predfinitepop**: Predictive Inference on Totals and Averages of Finite Populations Segmented in Planned and Unplanned Domains. Authors: Juan Carlos Martinez-Ovando, Sergio I. Olivares-Guzman, Adriana Roldan-Rodriguez.
- psData**: Download regularly maintained political science data sets and make commonly used, but infrequently updated variables based on this data. Author: Christopher Gandrud.
- pt**: Computational models for prospect theory and other theories of risky decision making. Author: Gary Au.
- pubmed.mineR**: Text mining of PubMed Abstracts. Authors: Jyoti Sharma, S. Ramachandran, Ab Rauf Shah. In view: *WebTechnologies*.
- pushoverr**: Send push notifications using Pushover. Author: Brian Connelly [aut, cre].
- qVarSel**: Variable Selection for Clustering and Classification. Author: Stefano Benati.
- qcr**: Quality control and reliability. Authors: Miguel Flores, Salvador Naya, Ruben Fernandez.
- qdapDictionaries**: Dictionaries and word lists for the **qdap** package. Author: Tyler Rinker.

- qdapTools**: Tools for the **qdap** package. Authors: Bryan Goodrich [ctb], Dason Kurkiewicz [ctb], Kirill Muller [ctb], Tyler Rinker [aut, cre].
- qlcMatrix**: Utility sparse matrix functions for Quantitative Language Comparison. Author: Michael Cysouw.
- qmethod**: Analysis of subjective perspectives using Q methodology. Author: Aiora Zabala [aut, cre].
- qqman**: Q-Q and manhattan plots for GWAS data. Author: Stephen Turner.
- quipu**: Summary charts of micro satellite profiles for a set of biological samples. Authors: Reinhard Simon, Pablo Carhuapoma, Vilma Hualla, Stef de Haan, Marc Ghislain, Jorge Nunez, Rene Gomez, Felipe de Mendiburu, William Roca, Merideth Bonierbale.
- r2d2**: Bivariate (Two-Dimensional) Confidence Region and Frequency Distribution. Authors: Arni Magnusson [aut], Julian Burgos [aut, cre], Gregory R. Warnes [ctb].
- rARPACK**: R wrapper of ARPACK for large scale eigenvalue/vector problems, on both dense and sparse matrices. Authors: Yixuan Qiu and authors of the ARPACK library. In view: *NumericalMathematics*.
- rAvis**: Interface to the bird-watching datasets at proyectoavis.com. Authors: Javier González Hernández, Sara Varela González.
- rClinicalCodes**: R tools for integrating with the www.clinicalcodes.org repository. Author: David Springate.
- rDVR**: Start, stop and save a video server from within R. Author: John Harrison [aut, cre].
- rHealthDataGov**: Retrieve data sets from the HealthData.gov data API. Author: Erin LeDell.
- rSCA**: Stepwise Cluster Analysis. Author: Xiuquan Wang.
- rTensor**: Tools for tensor analysis and decomposition. Authors: James Li and Jacob Bien and Martin Wells.
- rWBclimate**: Access World Bank climate data. Author: Edmund Hart [aut, cre].
- rag2ridges**: Ridge Estimation of Precision Matrices from High-Dimensional Data. Authors: Carel F.W. Peeters, Wessel N. van Wieringen.
- rainfreq**: Rainfall Frequency (Design Storm) Estimates from the US National Weather Service. Author: Gopi Goteti.
- randomUniformForest**: Random Uniform Forests for Classification and Regression. Author: Saip Ciss.
- randtests**: Testing randomness in R. Authors: Frederico Caeiro and Ayana Mateus.
- rapportools**: Miscellaneous (stats) helper functions with sane defaults for reporting. Authors: Aleksandar Blagoić and Gergely Daróczy.
- rareNMtests**: Ecological and biogeographical null model tests for comparing rarefaction curves. Authors: Luis Cayuela and Nicholas J. Gotelli.
- raters**: An index of inter-rater agreement among a set of raters. Authors: Daniele Giardiello, Piero Quatto and Stefano Vigliani.
- rbhl**: R interface to the Biodiversity Heritage Library. Authors: Scott Chamberlain [aut, cre], Karthik Ram [aut]. In view: *WebTechnologies*.
- rbison**: R interface to the USGS BISON API. Author: Scott Chamberlain [aut, cre]. In view: *WebTechnologies*.

- rbitcoinchartsapi**: R Package for the BitCoinCharts.com API. Author: Thomas P. Fuller.
- rdbrobust**: Robust data-driven statistical inference in Regression-Discontinuity designs. Authors: Sebastian Calonico, Matias D. Cattaneo, Rocio Titiunik.
- readODS**: Read ODS files and puts them into data frames. Author: Gerrit-Jan Schutten.
- recalls**: Access U.S. Federal Government Recall Data. Author: Thomas J. Leeper.
- regRSM**: Random Subspace Method (RSM) for linear regression. Authors: Pawel Teisseyre, Robert A. Klopotek.
- repfdr**: Replicability Analysis for Multiple Studies of High Dimension. Authors: Ruth Heller, Shachar Kaufman, Shay Yaacoby, Daniel Yekutieli.
- reportRx**: Tools for automatically generating reproducible clinical report. Authors: Ryan Del Bel, Wei Xu.
- resample**: Resampling functions. Author: Tim Hesterberg.
- resemble**: Regression and similarity evaluation for memory-based learning in spectral chemometrics. Authors: Leonardo Ramirez-Lopez and Antoine Stevens. In view: *ChemPhys*.
- revealedPrefs**: Revealed preferences and microeconomic rationality. Author: Julien Boelaert.
- rgabriel**: Gabriel Multiple Comparison Test and Plot the Confidence Interval on Barplot. Authors: Yihui Xie, Miao Yu.
- rgpui**: UI for the RGP genetic programming framework. Author: Oliver Flasch.
- rinat**: Access iNaturalist data through APIs. Authors: Vijay Barve, Edmund Hart.
- riverplot**: Sankey or ribbon plots. Author: January Weiner.
- rivervis**: River Visualisation Tool. Authors: Feng Mao, Yichuan Shi, and Keith Richards.
- rlist**: A toolset for working with lists. Author: Kun Ren.
- rmaf**: Refined Moving Average Filter. Author: Debin Qiu. In view: *TimeSeries*.
- rmatio**: Read and write matlab files. Authors: Stefan Widgren [aut, cre] (Author of the R interface to the C-library matio), Christopher Hulbert [aut] (Author of the C-library matio, <http://sourceforge.net/projects/matio/>). In view: *SpatioTemporal*.
- rmngb**: rmngb Miscellaneous. Author: Antoine Filipovic Pierucci.
- rnoaa**: NOAA climate data from R. Authors: Hart Edmund [aut, cre], Scott Chamberlain [aut], Karthik Ram [aut]. In view: *WebTechnologies*.
- robcor**: Robust Correlations. Author: Paul Smirnov. In view: *Robust*.
- robumeta**: Robust variance meta-regression. Authors: Zachary Fisher and Elizabeth Tipton. In views: *MetaAnalysis*, *Robust*.
- rorutadis**: Robust Ordinal Regression UTADIS. Author: Krzysztof Ciomek.
- rpart.utils**: Tools for parsing and manipulating rpart objects, including generating machine readable rules. Author: Craig Varrichio.
- rpartitions**: Code for integer partitioning. Authors: Ken Locey, Daniel McGlinn.
- rprintf**: Adaptive builder for formatted strings. Author: Kun Ren.
- rstudioapi**: Safely access the RStudio API. Author: RStudio.

- rvTDT**: Population control weighted rare-variants TDT. Authors: Yu Jiang, Andrew S. Allen.
- ryouready**: Companion to the “R your ready?” book. Author: Mark Heckmann.
- s2dverification**: Set of common tools for model diagnostics. Authors: Nicolau Manubens Gil, Virginie Guemas, Isabel Andreu-Burillo, Fabian Lienert, Javier Garcia-Serrano, Ludovic Auger.
- samplesize4surveys**: Sample size calculations for complex surveys. Author: Hugo Andres Gutierrez Rojas.
- sand**: Statistical Analysis of Network Data with R. Authors: Eric D Kolaczyk, Gabor Csardi.
- sbioPN**: Simulation of deterministic and stochastic spatial biochemical reaction networks using Petri Nets. Authors: Roberto Bertolusso and Marek Kimmel.
- scalreg**: Scaled sparse linear regression. Author: Tingni Sun.
- script**: Scrypt key derivation functions for R. Authors: RStudio, Inc.; Colin Percival.
- sdmvspecies**: Create virtual species for species distribution modelling. Authors: Xiaoquan Kong, Renyan Duan, Minyi Huang.
- seasonal**: R interface to X-13ARIMA-SEATS. Author: Christoph Sax. In views: *OfficialStatistics*, *TimeSeries*.
- seawaveQ**: U.S. Geological Survey seawaveQ model. Authors: Karen R. Ryberg and Aldo V. Vecchia.
- secrdesign**: Sampling Design for Spatially Explicit Capture–Recapture. Author: Murray Efford.
- semiArtificial**: Generator of semi-artificial data. Author: Marko Robnik-Sikonja.
- sensitivitymw**: Sensitivity analysis using weighted M-statistics. Author: Paul R. Rosenbaum.
- seqDesign**: Simulation and group sequential monitoring of randomized multi-arm two-stage Phase IIb/III treatment efficacy trials with time-to-event endpoints. Authors: Michal Juraska and Doug Grove, with contributions from Xuesong Yu and Peter B. Gilbert.
- sequenza**: Analysis and visualization of tumor genome sequencing data. Authors: Francesco Favero, Andrea M. Marquard, Tejal Joshi, Aron C. Eklund.
- sesem**: Spatially explicit structural equation modeling. Authors: Eric Lamb [aut, cre], Kerrie Mengersen [aut], Katherine Stewart [aut], Udayanga Attanayake [aut], Steven Siciliano [aut].
- sglOptim**: Sparse group lasso generic optimizer. Author: Martin Vincent.
- sglasso**: Lasso method for RCON(V,E) models. Author: Luigi Augugliaro.
- sglr**: Power and boundary calculations in pre-licensure vaccine trials using a sequential generalized likelihood ratio test. Authors: Balasubramanian Narasimhan [aut, cre], Mei-Chiung Shih [aut].
- sharpshootR**: A collection of functions to support soil survey. Author: USDA-NRCS Soil Survey Staff.
- shinyBS**: Twitter Bootstrap Components for Shiny. Author: Eric Bailey.
- shinyFiles**: A server-side file system viewer for **shiny**. Author: Thomas Lin Pedersen.

- showtext**: Enable (any) R Graphics Device to Show Text Using System Fonts. Authors: Yixuan Qiu and authors/contributors of the included software.
- shp2graph**: Convert a SpatialLinesDataFrame object to an “igraph-class” object. Author: Binbin Lu.
- shuffle**: The shuffle estimator for explainable variance. Author: Yuval Benjamini.
- siRSM**: Single-Index Response Surface Models. Authors: Huan Cheng, Mu Zhu.
- simplexreg**: Regression Analysis of Proportional Data Using Simplex Distribution. Authors: Peng Zhang, Zhengguo Qiu and Chengchun Shi.
- siplab**: Spatial individual-plant modelling. Author: Oscar Garcia. In view: *Spatial*.
- sitar**: SITAR growth curve analysis. Author: Tim Cole.
- skda**: Sparse (Multicategory) Kernel Discriminant Analysis. Authors: Len Stefanski, Yichao Wu, and Kyle White.
- slfm**: Tools for fitting sparse latent factor model. Authors: Joao Duarte and Vinicius Mayrink.
- soc.ca**: Specific correspondence analysis for the social sciences. Authors: Anton Grau Larsen, with contributions from Christoph Ellersgaard and Stefan Andrade.
- soilphysics**: Soil Physical Analysis. Authors: Anderson Rodrigo da Silva, Renato Paiva de Lima.
- solr**: General purpose R interface to Solr. Author: Scott Chamberlain [aut, cre].
- sortinghat**: Author: John A. Ramey.
- sotkanet**: Sotkanet R Tools. Authors: Leo Lahti, Einari Happonen, Juuso Parkkinen, Joona Lehtomaki.
- sp23design**: Design and Simulation of seamless Phase II-III Clinical Trials. Authors: Balasubramanian Narasimhan [aut, cre], Mei-Chiung Shih [aut], Pei He [aut].
- spBayesSurv**: Spatial Copula Modelling to Bayesian Survival Analysis. Authors: Haiming Zhou and Tim Hanson.
- spaMM**: Mixed models, particularly spatial GLMMs. Authors: François Rousset [aut, cre, cph], Jean-Baptiste Ferdy [aut, cph]. In view: *Spatial*.
- spatial.gev.bma**: Hierarchical spatial generalized extreme value (GEV) modeling with Bayesian Model Averaging (BMA). Author: Alex Lenkoski.
- spatialTailDep**: Estimation of spatial tail dependence models. Authors: Anna Kiriliouk, Johan Segers.
- spatsurv**: Bayesian spatial survival analysis with parametric proportional hazards models. Authors: Benjamin M. Taylor and Barry S. Rowlingson. In view: *Survival*.
- spcr**: Sparse principal component regression. Author: Shuichi Kawano.
- spdynmod**: Spatio-dynamic wetland plant communities model. Authors: Javier Martinez-Lopez, Babak Naimi, Julia Martinez-Fernandez.
- spectral.methods**: Singular Spectrum Analysis (SSA) tools for time series analysis. Author: Jannis v. Buttlar. In view: *TimeSeries*.
- spfrontier**: Spatial Stochastic Frontier models estimation. Author: Dmitry Pavlyuk.
- spocc**: R interface to many species occurrence data sources. Authors: Scott Chamberlain [aut, cre], Karthik Ram [aut], Ted Hart [aut]. In view: *WebTechnologies*.

- sqliter**: Connection wrapper to SQLite databases. Author: Wilson Freitas.
- ss3sim**: Fisheries stock assessment simulation testing with Stock Synthesis. Authors: Sean Anderson [aut, cre], Cole Monnahan [aut], Kelli Johnson [aut], Kotaro Ono [aut], Juan Valero [aut], Curry Cunningham [aut], Felipe Hurtado-Ferro [aut], Roberto Licandeo [aut], Carey McGilliard [aut], Melissa Muradian [ctb], Cody Szuwalski [aut], Katyana Vert-pre [aut], Athol Whitten [aut].
- sse**: Sample size estimation. Author: Thomas Fabbro.
- ssym**: Fitting Semiparametric Symmetric Regression Models. Authors: Luis Hernando Vanegas and Gilberto A. Paula.
- statfi**: statfi R tools. Authors: Leo Lahti, Juuso Parkkinen, Joona Lehtomaki.
- stilt**: Separable Gaussian Process Interpolation (Emulation). Authors: Roman Olson, Won Chang, Klaus Keller, and Murali Haran.
- stm**: Estimation of the Structural Topic Model. Authors: Margaret E. Roberts, Brandon M. Stewart and Dustin Tingley.
- stochprofML**: Stochastic Profiling using Maximum Likelihood Estimation. Author: Christiane Fuchs.
- stosim**: Stochastic Simulator for Reliability Modeling of Repairable Systems. Author: Jacob T. Ormerod.
- strap**: Stratigraphic Tree Analysis for Palaeontology. Author: Mark A. Bell Graeme T. Lloyd.
- streamMOA**: **stream** Interface to MOA Algorithms. Authors: Michael Hahsler [aut, cre, cph], Matthew Bolanos [aut, cph], John Forrest [aut, cph].
- stressr**: Fetch and plot financial stress index and component data. Author: Matt Barry.
- stringi**: Character string processing facilities. Authors: Marek Gagolewski and Bartek Tartanus (stringi source code); IBM and other contributors (ICU4C 52.1 source code); Unicode, Inc. (Unicode Character Database).
- stsm**: Structural Time Series Models. Author: Javier López-de-Lacalle. In view: [TimeSeries](#).
- stsm.class**: Class and Methods for Structural Time Series Models. Author: Javier López-de-Lacalle. In view: [TimeSeries](#).
- subsemble**: An Ensemble Method for Combining Subset-Specific Algorithm Fits. Authors: Erin LeDell, Stephanie Sapp, Mark van der Laan.
- survivalMPL**: Penalised Maximum Likelihood for Survival Analysis Models. Authors: Dominique-Laurent Couturier, Jun Ma, Stephane Heritier.
- sweSCB**: An R interface to the web API of Statistics Sweden. Authors: Love Hansson, Thomas Reinholdsson, Mans Magnusson. In view: [WebTechnologies](#).
- swirl**: Statistics with Interactive R Learning. Authors: Nick Carchedi [aut, cre], Bill Bauer [aut], Gina Grdina [aut], Sean Kross [aut].
- sybilccFBA**: Cost Constrained FLux Balance Analysis: MetabOlic Modeling with ENzyme kineTics (MOMENT). Author: Abdelmoneim Amer Desouki [aut, cre].
- sybilcycleFreeFlux**: cycle-Free Flux balance analysis: Efficient removal of thermodynamically infeasible cycles from metabolic flux distributions. Author: Abdelmoneim Amer Desouki [aut, cre].
- synchrony**: Methods for computing spatial, temporal, and spatiotemporal statistics. Author: Tarik C. Gouhier.

- synlik**: Synthetic Likelihood methods for intractable likelihoods. Authors: Matteo Fasiolo and Simon Wood.
- sysfonts**: Loading system fonts into R. Authors: Yixuan Qiu and authors/contributors of the included fonts.
- tab**: Functions for creating summary tables for statistical reports. Author: Dane R. Van Domelen.
- tableone**: Create “Table 1” to describe baseline characteristics. Authors: Kazuki Yoshida, Justin Bohn.
- threeboost**: Thresholded variable selection and prediction based on estimating equations. Authors: Julian Wolfson and Christopher Miller.
- tictoc**: Functions for timing R scripts, as well as implementations of Stack and List structures. Author: Sergei Izrailev.
- tigerstats**: R for Elementary Statistics. Authors: Rebekah Robinson and Homer White.
- tm.plugin.alceste**: Import texts from files in the Alceste format using the **tm** text mining framework. Author: Milan Bouchet-Valat [aut, cre]. In view: *NaturalLanguageProcessing*.
- tm.plugin.europresse**: Import articles from Europresse using the **tm** text mining framework. Author: Milan Bouchet-Valat [aut, cre]. In view: *NaturalLanguageProcessing*.
- tm.plugin.lexisnexis**: Import articles from LexisNexis using the **tm** text mining framework. Author: Milan Bouchet-Valat [aut, cre]. In view: *NaturalLanguageProcessing*.
- tmle.npvi**: Targeted Minimum Loss Estimation (TMLE) of a NP variable importance of a continuous exposure. Authors: Antoine Chambaz, Pierre Neuvial.
- toaster**: analytics and visualization with Aster Database. Author: Gregory Kanevsky. In view: *HighPerformanceComputing*.
- tosls**: Instrumental Variables Two Stage Least Squares estimation. Author: Zaghdoudi Taha.
- transnet**: Conducts transmission modeling on a bayesian network. Authors: Alun Thomas [aut, cph], Andrew Redd [com, cre, ctb].
- transport**: Optimal transport in various forms. Authors: Dominic Schuhmacher with substantial contributions of code by Bjoern Baehre and Carsten Gottschlich.
- tsintermittent**: Intermittent Time Series Forecasting. Authors: Nikolaos Kourentzes and Fotios Petropoulos. In view: *TimeSeries*.
- tsoutliers**: Automatic Detection of Outliers in Time Series. Author: Javier López-de-Lacalle. In view: *TimeSeries*.
- turfR**: TURF Analysis for R. Author: Jack Horne [aut, cre].
- tvrm**: Time Value of Money Functions. Author: Juan Manuel Truppia. In view: *Finance*.
- ucbthesis**: UC Berkeley graduate division thesis template. Author: Steven Pollack.
- unbalanced**: Implementation of different data-driven method for unbalanced datasets. Authors: Andrea Dal Pozzolo, Olivier Caelen and Gianluca Bontempi.
- uniReg**: Unimodal penalized spline regression using B-splines. Author: Claudia Koellmann.
- uplift**: Uplift Modeling. Author: Leo Guelman.
- userfriendlyscience**: Quantitative analysis made accessible. Author: Gjalt-Jorn Peters.

- ustyc:** Fetch US Treasury yield curve data. Author: Matt Barry.
- vardpoor:** Variance estimation for sample surveys by the ultimate cluster method. Authors: Juris Breidaks [aut, cre], Martins Liberts [aut].
- vbsr:** Variational Bayes Spike Regression Regularized Linear Models. Author: Benjamin Logsdon.
- vdg:** Variance Dispersion Graphs and Fraction of Design Space Plots. Author: Pieter Schoonees [aut, cre, cph].
- vecsets:** like base::sets tools but keeps duplicate elements. Author: Carl Witthoft.
- wbs:** Wild Binary Segmentation for multiple change-point detection. Authors: Rafal Baranowski and Piotr Fryzlewicz.
- weatherData:** Get Weather Data from the Web. Author: Ram Narasimhan. In view: *WebTechnologies*.
- wesanderson:** A Wes Anderson Palette Generator. Authors: Karthik Ram [aut, cre], Richards Clark [ctb].
- wrspathrow:** Functions for working with Worldwide Reference System (WRS). Author: Alex Zvoleff [aut, cre].
- wrspathrowData:** Data used by the **wrspathrow** package. Author: Alex Zvoleff [aut, cre].
- wskm:** Weighted k -means Clustering. Authors: Graham Williams [aut], Joshua Z Huang [aut], Xiaojun Chen [aut], Qiang Wang [aut], Longfei Xiao [aut], He Zhao [cre].
- wsrf:** Weighted Subspace Random Forest. Authors: Qinghan Meng [aut], He Zhao [aut, cre], Graham Williams [aut], Junchao Lv [ctb].
- wtrsk:** Competing risks regression models with covariate-adjusted censoring weight. Author: Peng He. In view: *Survival*.
- xergm:** Extensions for Exponential Random Graph Models (ERGM). Authors: Philip Leifeld [aut, cre], Skyler J. Cranmer [aut], Bruce A. Desmarais [aut].
- xhmmScripts:** XHMM R scripts. Author: Menachem Fromer.
- ycinterextra:** Yield curve or zero-coupon prices interpolation and extrapolation. Author: Thierry Moudiki. In view: *Finance*.
- ykmeans:** K-means using a target variable. Author: Yohei sato.
- yuima:** The YUIMA Project package for SDEs. Author: YUIMA Project Team.
- zCompositions:** Replacement of zeros and nondetects in compositional data sets. Authors: Javier Palarea-Albaladejo and Josep Antoni Martin-Fernandez.
- zoib:** Bayesian Inference for Zero/One Inflated Beta Regression. Author: Fang Liu with contributions YunChuan Kong.

Other changes

The following packages were moved to the Archive: ANN, BINCO, BayesTree, CAscaling, CGP, CalciOMatic, ChangeAnomalyDetection, ClinicalRobustPriors, CvM2SL1Test, CvM2SL2Test, DPM.GGM, DataFrameConstr, DesignPatterns, EasyUpliftTree, EstSimPDMP, ExactSampling, FunNet, GGIR, GGally, GOFSN, GUTS, GridR, HGLMMM, HIBAG, HMMmix, HiDimDA, JGR, JMLSD, LearnEDA, MAVTgsa, MCUSUM, MEWMA, MLDA, MTSKNN, MetaPath, MultiChIPmixHMM, PL.popN, POT, Peak2Trough, QLSpline, QT, RLastFM, RSearchYJ, RSofia, RTDAmeritrade,

RTisean, RWebMA, RadialPlotter, RcppZiggurat, RecordLinkage, RiDMC, Rvelslant, SDisc, SNPRelate, SRMA, SigWinR, SlimPLS, Snowball, SparseTSCGM, SpatialPack, StochaTR, StructR, TAHMMAnnot, TGUICore, TGUITeaching, TPAM, TSPC, TWIX, TinnR, ToxLim, VisCov, VizCompX, WMTregions, WeightedCluster, XiMpLe, YjdnJlp, adk, ageprior, aggrisk, binarySimCLF, birch, bscr, caGUI, cems, charlson, cladoRpp, cldr, colcor, compositionsGUI, conveyol, copas, corrperm, cpm, dynpred, easystab, eco, ecp, eigendog, el.convex, epibase, evtree, flubase, gamesNws, gazetools, gdsfmt, gearman, glmmAK, glrt, gmvalid, govdat, gppois, gregmisc, h5r, hints, hof, iSubpathwayMiner, iWebPlots, imputation, indicpecies, int64, kernelPop, knorm, koRpus, leiv, ljr, lossDev, magnets, mchof, mixfdr, mtcreator, mugnet, mutatr, ndvits, nga, nloptwrap, numConversion, odprism, opencpu.encode, orth, pa, pamctdp, parboost, pcrsim, pcurve, pgmm, postgwas, ppMeasures, probemapper, protoclust, pycno, rHadoopClient, rbamtools, readMETEO, rlandscape, rmac, rngSetSeed, rolasized, ropensnp, scapeMCMC, sensitivityPStrat, seq2R, sessionTools, shapes, sheldusr, sifds, simPopulation, sinatra, skewtools, snort, spclust, sprint, spsir, stochmod, stratasphere, survivalBIV, tagcloud, tclust, tmg, tonymisc, ttime, validator, vmv, waveclock, websockets, wethepeople, xpose4classic, xpose4data, xpose4generic, xpose4specific

The following packages were resurrected from the Archive: [BootPR](#), [CatDyn](#), [LSC](#), [RFreak](#), [RHive](#), [RcmdrPlugin.pointG](#), [RcmdrPlugin.sos](#), [Sim.DiffProc](#), [anchors](#), [binhf](#), [binomSamSize](#), [bujar](#), [cem](#), [cladoRcpp](#), [clues](#), [clusterfly](#), [cond](#), [cplm](#), [csampling](#), [dtc](#), [elrm](#), [epinet](#), [extfunnel](#), [fbati](#), [financial](#), [fitDRC](#), [fractal](#), [geospacom](#), [hydrogeo](#), [languageR](#), [ltsk](#), [marg](#), [mblm](#), [mmeta](#), [nlreg](#), [npst](#), [pcaPA](#), [polytomous](#), [ripa](#), [rsdepth](#), [simone](#), [spacom](#), [tikzDevice](#)

The following package had to be removed: [CDS](#).

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

Achim Zeileis
Universität Innsbruck, Austria
Achim.Zeileis@R-project.org

Changes in R

From 3.0.3 to 3.1.1

by the R Core Team

CHANGES IN R 3.1.1

NEW FEATURES

- When `attach()` reports conflicts, it does so compatibly with `library()` by using `message()`.
- R CMD Sweave no longer cleans any files by default, compatibly with versions of R prior to 3.1.0. There are new options `'--clean'`, `'--clean=default'` and `'--clean=keepOuts'`.
- `tools::buildVignette()` and `tools::buildVignettes()` with `clean = FALSE` no longer remove any created files. `buildvignette()` gains a `keep` argument for more cleaning customization.
- The Bioconductor 'version' used by `setRepositories()` can now be set by environment variable `R_BIOC_VERSION` at runtime, not just when R is installed. (It has been stated that Bioconductor will switch from 'version' 2.14 to 'version' 3.0 during the lifetime of the R 3.1 series.)
- Error messages from bugs in embedded 'Sexpr' code in Sweave documents now report the source location.
- `type.convert()`, `read.table()` and similar `read.*()` functions get a new `numerals` argument, specifying how numeric input is converted when its conversion to double precision loses accuracy. The default value, `"allow.loss"` allows accuracy loss, as in R versions before 3.1.0.
- For some compilers, integer addition could overflow without a warning. R's internal code for both integer addition and subtraction is more robust now. (PR#15774)
- The function determining the default number of knots for `smooth.spline()` is now exported, as `.nknots.smspl()`.
- `dbeta()`, `pbeta()`, `qbeta()` and `rbeta()` are now defined also for $a = 0$, $b = 0$, or infinite a and b (where they typically returned NaN before).
- Many package authors report that the RStudio graphics device does not work correctly with their package's use of `dev.new()`. The new option `dev.new(noRStudioGD = TRUE)` replaces the RStudio override by the default device as selected by R itself, still respecting environment variables `R_INTERACTIVE_DEVICE` and `R_DEFAULT_DEVICE`.
- `readRDS()` now returns visibly.
- Modifying internal logical scalar constants now results in an error instead of a warning.
- `install.packages(repos = NULL)` now accepts `http://` or `ftp://` URLs of package archives as well as file paths, and will download as required. In most cases `repos = NULL` can be deduced from the extension of the URL.
- The warning when using partial matching with the `$` operator on data frames is now only given when `options("warnPartialMatchDollar")` is `TRUE`.
- Package help requests like `package?foo` now try the package `foo` whether loaded or not.

- General help requests now default to trying all loaded packages, not just those on the search path.
- Added a new function `promptImport()`, to generate a help page for a function that was imported from another package (and presumably re-exported, or help would not be needed).

INSTALLATION and INCLUDED SOFTWARE

- configure option `'--with-internal-tzcode'` can now be used with variable `rsharedir`.
- The included version of PCRE has been updated to 8.35.
- There is a new target `make uninstall-libR` to remove an installed shared/static `'libR'`. `make install-libR` now works if a sub-architecture is used, although the user will need to specify `libdir` differently for different sub-architectures.
- There is more extensive advice on which LaTeX packages are required to install R or to make package manuals (as done by `R CMD check`) in the 'Writing R Extensions' manual.
- Compilers/linkers were handling the visibility control in `'src/extra/xz'` inconsistently (and apparently in some cases incorrectly), so it has been simplified. (PR#15327)
- (Windows) There is updated support for the use of ICU for collation: see the 'R Installation and Administration Manual'.

BUG FIXES

- `dbinom(x,n)`, `pbinom()`, `dpois()`, etc, are slightly less restrictive in checking if `n` is integer-valued. (Wish of PR#15734.)
- `pchisq(x,df,ncp,log.p = TRUE)` is more accurate and no longer underflows for small `x` and `ncp < 80`, e.g, for `pchisq(1e-5,df = 100,ncp = 1,log = TRUE)`. (Based on PR#15635 and a suggestion by Roby Joehanes.)
- The `s` ("step into") command in the debugger would cause R to step into expressions evaluated there, not just into functions being debugged. (PR#15770)
- The C code used by `strptime()` rejected time-zone offsets of more than +1200 (+1245, +1300 and +1400 can occur). (PR#15768)
- (Windows only.) `png(type = "cairo",antialias = "gray")` was not accepted. (PR#15760)
- Use of `save(...,envir=)` with named objects could fail. (PR#15758)
- `Sweave()` mis-parsed `'Sexpr'` expressions that contained backslashes. (PR#15779)
- The return value from `options(foo = NULL)` was not the previous value of the option. (PR#15781)
- `enc2utf8()` and `enc2native()` did not always mark the encoding of the return values when it was known.
- `dnbinom(x,size = <large>,mu,log = TRUE)` no longer underflows to `-Inf` for large `mu`, thanks to a suggestion from Alessandro Mammana (MPI MolGen, Berlin).
- `pbeta(x,a,b,log = TRUE)` no longer behaves discontinuously (in a small `x`-region) because of denormalized numbers. Also, `pbeta(1-1e-12,1e30,1.001,log=TRUE)` now terminates "in real time".

- The "CRAN" filter (see `available.packages()`) no longer removes duplicates other than of packages on CRAN, and does not fail if there is no CRAN repository in `getOption("repos")`.
- The device listing from `dev2bitmap()` and `bitmap()` was truncated to 1000 characters: modern versions of GhostScript on most platforms have many more devices.
- (Windows.) Commands such as `Sys.which()` and `pipe()` which needed to find the full path to a command could segfault if the 'long' path name was much longer than the 'short' path name (which `Sys.which()` returns), as the behaviour of the Windows API call had changed.
- R CMD build will fail with an error if one of the packages specified in the 'VignetteBuilder' field is not installed. (Without loading those packages it cannot be ascertained which files are intended to be vignettes. This means that the 'VignetteBuilder' packages have to be installed for package checking too.) (Wish of PR#15775.)
- Misguided attempts to use `chull()` with non-finite points now give an error (related to PR#15777).
- For a formula with exactly 32 variables the 32nd variable was aliased to the intercept in some C-level computations of terms, so that for example attempting to remove it would remove the intercept instead (and leave a corrupt internal structure). (PR#15735)
- `anyDuplicated()` silently returned wrong values when the first duplicate was at an index which was too large to be stored in an integer vector (although a lot of RAM and patience would have been needed to encounter this).
- `tools::Rd2ex(commentDontrun = FALSE)` failed if the block had only one line.
- Hexadecimal constants such as `0x110p-5L` which were incorrectly qualified by `L` were parsed incorrectly since R 3.0.0, with a slightly garbled warning. (PR#15753)
- `system()` returned success on some platforms even if the system was unable to launch a process. (PR#15796)
- (Windows Rgui console.) Unbuffered output was sometimes not output immediately if the prompt was not on the last line of the console.
- The built-in help server did not declare the encoding for the 'DESCRIPTION' or other text files to be the package encoding, so non-ASCII characters could be displayed incorrectly.
- R is now trying harder to not cleanup child processes that were not spawned by `mcpipeline()` on platforms that provide information about the source process of the SIGCHLD signal. This allows 3rd party libraries to manage the exit status of children that they spawn without R interfering.
- `mcmapply()` was only parallelizing if the number of jobs was bigger than the number of cores. It now parallelizes if the number of jobs is more than one.
- Auto-printing would re-evaluate its argument when trying to dispatch to a print method. This is now avoided when possible.
- Unserializing (including `load()` and `readRDS()`) could silently return incorrect numeric values from ASCII saves if there was a read error.
- `getParseData()` could return incorrect values for the parents of some elements. (Reported by Andrew Redd.)
- Attempting to use data frames of 2^{31} or more rows with `merge()` or to create a merged data frame of that size now gives a clearer error message.

- `parse()` did not check its file argument was a connection if it was not a character string, so e.g. `parse(FALSE)` attempted to read from `stdin`.
Nor did `dump()` and `dput()`.
- The "help.try.all.packages" option was ignored when the shortcut syntax for help was used, e.g. `?foo`.
- A potential segfault in string allocation has been fixed. (Found by Radford Neal.)
- Potential memory protection errors in `sort()` and `D()` have been fixed. (Found by Radford Neal.)
- Fixed a lack of error checking in graphics event functions. (Found by Radford Neal; a different patch used here than the one in `pqR`.)
- `numericDeriv()` sometimes miscalculated the gradient. (PR#15849, reported originally by Radford Neal)

CHANGES IN R 3.1.0

NEW FEATURES

- `type.convert()` (and hence by default `read.table()`) returns a character vector or factor when representing a numeric input as a double would lose accuracy. Similarly for complex inputs.
If a file contains numeric data with unrepresentable numbers of decimal places that are intended to be read as numeric, specify `colClasses` in `read.table()` to be "numeric".
- `tools::Rdiff(useDiff = FALSE)` is closer to the POSIX definition of `diff -b` (as distinct from the description in the man pages of most systems).
- New function `anyNA()`, a version of `any(is.na(.))` which is fast for atomic vectors, based on a proposal by Tim Hesterberg. (Wish of PR#15239.)
- `arrayInd(*, useNames = TRUE)` and, analogously, `which(*, arr.ind = TRUE)` now make use of `names(.dimnames)` when available.
- `is.unsorted()` now also works for raw vectors.
- The "table" method for `as.data.frame()` (also useful as `as.data.frame.table()`) now passes `sep` and `base` arguments to `provideDimnames()`.
- `uniroot()` gets new optional arguments, notably `extendInt`, allowing to auto-extend the search interval when needed. The return value has an extra component, `init.it`.
- `switch(f, ...)` now warns when `f` is a factor, as this typically happens accidentally where the user meant to pass a character string, but `f` is treated as integer (as always documented).
- The parser has been modified to use less memory.
- The way the unary operators `(+ -!)` handle attributes is now more consistent. If there is no coercion, all attributes (including class) are copied from the input to the result: otherwise only names, dims and dimnames are.
- `colorRamp()` and `colorRampPalette()` now allow non-opaque colours and a ramp in opacity via the new argument `alpha = TRUE`. (Suggested by Alberto Krone-Martins, but optionally as there are existing uses which expect only RGB values.)
- `grid.show.layout()` and `grid.show.viewport()` get an optional `vp.ex` argument.

- There is a new function `find_gs_cmd()` in the **tools** package to locate a GhostScript executable. (This is an enhanced version of a previously internal function there.)
- `object.size()` gains a `format()` method.
- There is a new family, "ArialMT", for the `pdf()` and `postscript()` devices. This will only be rendered correctly on viewers which have access to Monotype TrueType fonts (which are sometimes requested by journals).
- The text and PDF news files, including 'NEWS' and 'NEWS.2', have been moved to the 'doc' directory.
- `combn(x, simplify = TRUE)` now gives a factor result for factor input `x` (previously user error). (Related to PR#15442.)
- Added `utils::fileSnapshot()` and `utils::changedFiles()` functions to allow snapshots and comparison of directories of files.
- `make.names(names, unique=TRUE)` now tries to preserve existing names. (Suggestion of PR#15452.)
- New functions `cospi(x)`, `sinpi(x)`, and `tanpi(x)`, for more accurate computation of `cos(pi*x)`, etc, both in R and the C API. Using these gains accuracy in some cases, e.g., inside `lgamma()` or `besselI()`. (Suggested by Morten Welinder in PR#15529.)
- `print.table(x, zero.print = ".")` now also has an effect when `x` is not integer-valued.
- There is more support to explore the system's idea of time-zone names. `Sys.timezone()` tries to give the current system setting by name (and succeeds at least on Linux, OS X, Solaris and Windows), and `OlsonNames()` lists the names in the system's Olson database. `Sys.timezone(location = FALSE)` gives the previous behaviour.
- Platforms with a 64-bit `time_t` type are allowed to handle conversions between the "POSIXct" and "POSIXlt" classes for date-times outside the 32-bit range (before 1902 or after 2037): the existing workarounds are used on other platforms. (Note that time-zone information for post-2037 is speculative at best, and the OS services are tested for known errors and so not used on OS X.)
Currently `time_t` is usually long and hence 64-bit on Unix-alike 64-bit platforms: however in several cases the time-zone database is 32-bit. For R for Windows it is 64-bit (for both architectures as from this version).
- The "save.defaults" option can include a value for `compression_level`. (Wish of PR#15579.)
- `colSums()` and friends now have support for arrays and data-frame columns with 2^{31} or more elements.
- `as.factor()` is faster when `f` is an unclassed integer vector (for example, when called from `tapply()`).
- `fft()` now works with longer inputs, from the 12 million previously supported up to 2 billion. (PR#15593)
- Complex `svd()` now uses LAPACK subroutine ZGESDD, the complex analogue of the routine used for the real case.
- Sweave now outputs '.tex' files in UTF-8 if the input encoding is declared to be UTF-8, regardless of the local encoding. The UTF-8 encoding may now be declared using a LaTeX comment containing the string `%\SweaveUTF8` on a line by itself.
- `file.copy()` gains a `copy.date` argument.

- Printing of date-times will make use of the time-zone abbreviation in use at the time, if known. For example, for Paris pre-1940 this could be 'LMT', 'PMT', 'WET' or 'WEST'. To enable this, the "POSIXlt" class has an optional component "zone" recording the abbreviation for each element.

For platforms which support it, there is also a component "gmtoff" recording the offset from GMT where known.

- (On Windows, by default on OS X and optionally elsewhere.) The system C function `strftime` has been replaced by a more comprehensive version with closer conformance to the POSIX 2008 standard.
- `dnorm(x, log = FALSE)` is more accurate (but somewhat slower) for $|x| > 5$; as suggested in PR#15620.
- Some versions of the `tiff()` device have further compression options.
- `read.table()`, `readLines()` and `scan()` have a new argument to influence the treatment of embedded nuls.
- Avoid duplicating the right hand side values in complex assignments when possible. This reduces copying of replacement values in expressions such as `Z$a <-a0` and `ans[[i]] <-tmp`: some package code has relied on there being copies.

Also, a number of other changes to reduce copying of objects; all contributed by or based on suggestions by Michael Lawrence.

- The `fast` argument of `KalmanLike()`, `KalmanRun()` and `KalmanForecast()` has been replaced by `update`, which instead of updating `mod` in place, optionally returns the updated model in an attribute "mod" of the return value.
- `arima()` and `makeARIMA()` get a new optional argument `SSinit`, allowing the choice of a different state space initialization which has been observed to be more reliable close to non-stationarity: see PR#14682.
- `warning()` has a new argument `noBreaks.`, to simplify post-processing of output with `options(warn = 1)`.
- `pushBack()` gains an argument `encoding`, to support reading of UTF-8 characters using `scan()`, `read.table()` and related functions in a non-UTF-8 locale.
- `all.equal.list()` gets a new argument `use.names` which by default labels differing components by names (if they match) rather than by integer index. Saved R output in packages may need to be updated.
- The methods for `all.equal()` and `attr.all.equal()` now have argument `check.attributes` after `...` so it cannot be partially nor positionally matched (as it has been, unintentionally).

A side effect is that some previously undetected errors of passing empty arguments (no object between commas) to `all.equal()` are detected and reported.

There are explicit checks that `check.attributes` is logical, `tolerance` is numeric and `scale` is `NULL` or numeric. This catches some unintended positional matching.

The message for `all.equal.numeric()` reports a "scaled difference" only for `scale != 1`.

- `all.equal()` now has a "POSIXt" method replacing the "POSIXct" method.
- The "Date" and "POSIXt" methods of `seq()` allows `by = "quarter"` for completeness (by = "3 months" always worked).

- `file.path()` removes any trailing separator on Windows, where they are invalid (although sometimes accepted). This is intended to enhance the portability of code written by those using POSIX file systems (where a trailing `/` can be used to confine path matching to directories).
- New function `agrep1()` which like `grep1()` returns a logical vector.
- `fifo()` is now supported on Windows. (PR#15600)
- `sort.list(method = "radix")` now allows negative integers (wish of PR#15644).
- Some functionality of `print.ts()` is now available in `.preformat.ts()` for more modularity.
- `mcparallel()` gains an option `detach = TRUE` which allows execution of code independently of the current session. It is based on a new `estranged = TRUE` argument to `mcfork()` which forks child processes such that they become independent of the parent process.
- The `pdf()` device omits circles and text at extremely small sizes, since some viewers were failing on such files.
- The rightmost break for the "months", "quarters" and "years" cases of `hist.POSIXlt()` has been increased by a day. (Inter alia, fixes PR#15717.)
- The handling of `DF[i,] <-a` where `i` is of length 0 is improved. (Inter alia, fixes PR#15718.)
- `hclust()` gains a new method "ward.D2" which implements Ward's method correctly. The previous "ward" method is "ward.D" now, with the old name still working. Thanks to research and proposals by Pierre Legendre.
- The `sunspot.month` dataset has been amended and updated from the official source, whereas the `sunspots` and `sunspot.year` datasets will remain immutable. The documentation and source links have been updated correspondingly.
- The `summary()` method for "lm" fits warns if the fit is essentially perfect, as most of the summary may be computed inaccurately (and with platform-dependent values). Programmers who use `summary()` in order to extract just a component which will be reliable (e.g. `$cov.unscaled`) should wrap their calls in `suppressWarnings()`.

INSTALLATION and INCLUDED SOFTWARE

- The included version of LAPACK has been updated to 3.5.0.
- There is some support for parallel testing of an installation, by setting `TEST_MC_CORES` to an integer greater than one to indicate the maximum number of cores to be used in parallel. (It is worth specifying at least 8 cores if available.) Most of these require a make program (such as GNU make and dmake) which supports the `$MAKE -j nproc` syntax.

Except on Windows: the tests of standard package examples in `make check` are done in parallel. This also applies to running `tools::testInstalledPackages()`.

The more time-consuming regression tests are done in parallel.

The package checks in `make check-devel` and `make check-recommended` are done in parallel.

- More of `make check` will work if recommended packages are not installed: but recommended packages remain needed for thorough checking of an R build.
- The version of 'tzcode' included in 'src/extra/tzone' has been updated. (Formerly used only on Windows.)

- The included (64-bit) time-zone conversion code and Olson time-zone database can be used instead of the system version: use configure option ‘--with-internal-tzcode’. This is the default on Windows and OS X. (Note that this does not currently work if a non-default rsharedir configure variable is used.)

(It might be necessary to set environment variable TZ on OSes where this is not already set, although the system timezone is deduced correctly on at least Linux, OS X and Windows.)

This option also switches to the version of strftime included in directory ‘src/extra/tzone’.

- configure now tests for a C++11-compliant compiler by testing some basic features. This by default tries flags for the compiler specified by ‘CXX’, but an alternative compiler, options and standard can be specified by variables ‘CXX1X’, ‘CXX1XFLAGS’ and ‘CXX1XSTD’ (e.g. ‘-std=gnu++11’).
- R can now optionally be compiled to use reference counting instead of the NAMED mechanism by defining SWITCH_TO_REFCNT in ‘Rinternals.h’. This may become the default in the future.
- There is a new option ‘--use-system-tre’ to use a suitable system **tre** library: at present this means a version from their git repository, after corrections. (Wish of PR#15660.)

PACKAGE INSTALLATION

- The CRANextra repository is no longer a default repository on Windows: all the binary versions of packages from CRAN are now on CRAN, although CRANextra contains packages from Omegahat and elsewhere used by CRAN packages.
- Only vignettes sources in directory ‘vignettes’ are considered to be vignettes and hence indexed as such.
- In the ‘DESCRIPTION’ file,

License: X11

is no longer recognized as valid. Use ‘MIT’ or ‘BSD_2_clause’ instead, both of which need ‘+ file LICENSE’.

- For consistency, entries in ‘.Rinstignore’ are now matched case-insensitively on all platforms.
- Help for S4 methods with very long signatures now tries harder to split the description in the ‘Usage’ field to no more than 80 characters per line (some packages had over 120 characters).
- R CMD INSTALL --build (not Windows) now defaults to the internal tar() unless R_INSTALL_TAR is set.
- There is support for compiling C++11 code in packages on suitable platforms: see ‘Writing R Extensions’.
- Fake installs now install the contents of directory ‘inst’: some packages use this to install e.g. C++ headers for use by other packages that are independent of the package itself. Option ‘--no-inst’ can be used to get the previous behaviour.

DEBUGGING

- The behaviour of the code browser has been made more consistent, in part following the suggestions in PR#14985.
- Calls to `browser()` are now consistent with calls to the browser triggered by `debug()`, in that Enter will default to `n` rather than `c`.
- A new browser command `s` has been added, to “step into” function calls.
- A new browser command `f` has been added, to “finish” the current loop or function.
- Within the browser, the command `help` will display a short list of available commands.

UTILITIES

- Only vignettes sources in directory ‘vignettes’ are considered to be vignettes by `R CMD check`. That has been the preferred location since R 2.14.0 and is now obligatory.
- For consistency, `R CMD build` now matches entries in ‘.Rbuildignore’ and ‘vignettes/.install_extras’ case-insensitively on all platforms (not just on Windows).
- `checkFF()` (called by `R CMD check` by default) can optionally check foreign function calls for consistency with the registered type and argument count. This is the default for `R CMD check --as-cran` or can be enabled by setting environment variable `_R_CHECK_FF_CALLS_` to ‘registration’ (but is in any case suppressed by ‘--install=no’). Because this checks calls in which `.NAME` is an R object and not just a literal character string, some other problems are detected for such calls.

Functions `suppressForeignCheck()` and `dontCheck()` have been added to allow package authors to suppress false positive reports.

- `R CMD check --as-cran` warns about a false value of the ‘DESCRIPTION’ field ‘BuildVignettes’ for Open Source packages, and ignores it. (An Open Source package needs to have complete sources for its vignettes which should be usable on a suitably well-equipped system).
- `R CMD check --no-rebuild-vignettes` is defunct:
`R CMD check --no-build-vignettes` has been preferred since R 3.0.0.
- `R CMD build --no-vignettes` is defunct:
`R CMD build --no-build-vignettes` has been preferred since R 3.0.0.
- `R CMD Sweave` and `R CMD Stangle` now process both Sweave and non-Sweave vignettes. The `tools::buildVignette()` function has been added to do the same tasks from within R.
- The flags returned by `R CMD config --ldflags` and (where installed) `pkg-config --libs libR` are now those needed to link a front-end against the (shared or static) R library.
- ‘Sweave.sty’ has a new option ‘[inconsolata]’.
- `R CMD check` customizations such as `_R_CHECK_DEPENDS_ONLY_` make available packages only in ‘LinkingTo’ only for installation, and not for loading/runtime tests.
- `tools::checkFF()` reports on `.C` and `.Fortran` calls with `DUP = FALSE` if argument `check_DUP` is true. This is selected by `R CMD check` by default.
- `R CMD check --use-gct` can be tuned to garbage-collect less frequently using `gctorture2()` via the setting of environment variable `_R_CHECK_GCT_N_`.
- Where supported, `tools::texi2dvi()` limits the number of passes tried to 20.

C-LEVEL FACILITIES

- (Windows only) A function `R_WaitEvent()` has been added (with declaration in header 'R.h') to block execution until the next event is received by R.
- Remapping in the 'Rmath.h' header can be suppressed by defining 'R_NO_REMAP_RMATH'.
- The remapping of `rround()` in header 'Rmath.h' has been removed: use `fround()` instead.
- `ftrunc()` in header 'Rmath.h' is now a wrapper for the C99 function `trunc()`, which might as well be used in C code: `ftrunc()` is still needed for portable C++ code.
- The never-documented remapping of `prec()` to `fprec()` in header 'Rmath.h' has been removed.
- The included LAPACK subset now contains ZGESDD and ZGELSD.
- The function `LENGTH()` now checks that it is only applied to vector arguments. However, in packages `length()` should be used. (In R itself `LENGTH()` is a macro without the function overhead of `length()`.)
- Calls to `SET_VECTOR_ELT()` and `SET_STRING_ELT()` are now checked for indices which are in-range: several packages were writing one element beyond the allocated length.
- `allocVector3` has been added which allows custom allocators to be used for individual vector allocations.

DEPRECATED AND DEFUNCT

- `chol(pivot = TRUE, LINPACK = TRUE)` is defunct.
Arguments `EISPACK` for `eigen()` and `LINPACK` for `chol()`, `chol2inv()`, `solve()` and `svd()` are ignored: LAPACK is always used.
- `.find.package()` and `.path.package()` are defunct: only the versions without the initial dot introduced in R 2.13.0 have ever been in the API.
- Partial matching when using the `$` operator *on data frames* now throws a warning and may become defunct in the future. If partial matching is intended, replace `foo$bar` by `foo[["bar", exact = FALSE]]`.
- The long-deprecated use of `\synopsis` in the 'Usage' section of '.Rd' files has been removed: such sections are now ignored (with a warning).
- `package.skeleton()`'s deprecated argument `namespace` has been removed.
- Many methods are no longer exported by package **stats**. They are all registered on their generic, which should be called rather than calling a method directly.
- Functions `readNEWS()` and `checkNEWS()` in package **tools** are defunct.
- `download.file(method = "lynx")` is deprecated.
- `.C(DUP = FALSE)` and `.Fortran(DUP = FALSE)` are now deprecated, and may be disabled in future versions of R. As their help has long said, `.Call()` is much preferred. R CMD check notes such usages (by default).
- The workaround of setting `R_OSX_VALGRIND` has been removed: it is not needed in current valgrind.

BUG FIXES

- Calling `lm.wfit()` with no non-zero weights gave an array-overflow in the Fortran code and a not very sensible answer. It is now special-cased with a simpler answer (no qr component).
- Error messages involving non-syntactic names (e.g. as produced by ``\r`` when that object does not exist) now encode the control characters. (Reported by Hadley Wickham.)
- `getGraphicsEvent()` caused 100% usage of one CPU in Windows. (PR#15500)
- `nls()` with no `start` argument may now work inside another function (scoping issue).
- `pbeta()` and similar work better for very large (billions) `ncp`.
- Where time zones have changed abbreviations over the years, the software tries to more consistently use the abbreviation appropriate to the time or if that is unknown, the current abbreviation. On some platforms where the C function `localtime` changed the `tzname` variables the reported abbreviation could have been that of the last time converted.
- `all.equal(list(1), identity)` now works.
- Bug fix for pushing viewports in **grid** (reported by JJ Allaire and Kevin Ushey).
NOTE for anyone poking around within the graphics engine display list (despite the warnings not to) that this changes what is recorded by **grid** on the graphics engine display list.
- Extra checks have been added for unit resolution and conversion in **grid**, to catch instances of division-by-zero. This may introduce error messages in existing code and/or produce a different result in existing code (but only where a non-finite location or dimension may now become zero).
- Some bugs in TRE have been corrected by updating from the git repository. This allows R to be installed on some platforms for which this was a blocker (PR#15087 suggests Linux on ARM and HP-UX).
- `?` applied to a call to an S4 generic failed in several cases. (PR#15680)
- The implicit S4 generics for primitives with `...` in their argument list were incorrect. (PR#15690)
- Bug fixes to `methods::callGeneric()`. (PR#15691)
- The bug fix to `aggregate()` in PR#15004 introduced a new bug in the case of no grouping variables. (PR#15699)
- In rare cases printing deeply nested lists overran a buffer by one byte and on a few platforms segfaulted. (PR#15679)
- The dendrogram method of `as.dendrogram()` was hidden accidentally, (PR#15703), and `order.dendrogram(d)` gave too much for a leaf `d`. (PR#15702)
- R would try to kill processes on exit that have pids ever used by a child process spawned by `mcpParallel` even though the current process with that pid was not actually its child.
- `cophenetic()` applied to a "dendrogram" object sometimes incorrectly returned a "Labels" attribute with dimensions. (PR#15706)
- `printCoefmat()` called from quite a few `print()` methods now obeys small `getOption("width")` settings, line wrapping the "signif. codes" legend appropriately. (PR#15708)

- `model.matrix()` assumed that the stored dimnames for a matrix was NULL or length 2, but length 1 occurred.
- The clipping region for a device was sometimes used in base graphics before it was set.

CHANGES IN R 3.0.3

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.)
- On a Unix-alike `Sys.timezone()` returns NA if the environment variable TZ is unset, to distinguish it from an empty string which on some OSes means the ‘UTC’ time zone.
- The backtick may now be escaped in strings, to allow names containing them to be constructed, e.g. ‘` `’. (PR#15621)
- `read.table()`, `readLines()` and `scan()` now warn when an embedded nul is found in the input. (Related to PR#15625 which was puzzled by the behaviour in this unsupported case.)
- (Windows only.) `file.symlink()` works around the undocumented restriction of the Windows system call to backslashes. (Wish of PR#15631.)
- `KalmanForecast(fast = FALSE)` is now the default, and the help contains an example of how `fast = TRUE` can be used in this version. (The usage will change in 3.1.0.)
- `strptime()` now checks the locale only when locale-specific formats are used and caches the locale in use: this can halve the time taken on OSes with slow system functions (e.g. OS X).
- `strptime()` and the `format()` methods for classes “POSIXct”, “POSIXlt” and “Date” recognize strings with marked encodings: this allows, for example, UTF-8 French month names to be read on (French) Windows.
- `iconv(to = "utf8")` is now accepted on all platforms (some implementations did already, but GNU **libiconv** did not: however converted strings were not marked as being in UTF-8). The official name, “UTF-8” is still preferred.
- `available.packages()` is better protected against corrupt metadata files. (A recurring problem with Debian package **shogun-r**: PR#14713.)
- Finalizers are marked to be run at garbage collection, but run only at a somewhat safer later time (when interrupts are checked). This circumvents some problems with finalizers running arbitrary code during garbage collection (the known instances being `running.options()` and (C-level) `path.expand()` re-entrantly).

INSTALLATION and INCLUDED SOFTWARE

- The included version of PCRE has been updated to 8.34. This fixes bugs and makes the behaviour closer to Perl 5.18. In particular, the concept of ‘space’ includes ‘VT’ and hence agrees with POSIX’s.

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.
- `install.packages(dependencies = value)` for `value = NA` (the default) or `value = TRUE` omits packages only in LinkingTo for binary package installs.

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’.

- The code for date-times before 1902 assumed that the offset from GMT in 1902 was a whole number of minutes: that was not true of Paris (as recorded on some platforms).
- Using `Sys.setlocale` to set `LC_NUMERIC` to "C" (to restore the sane behavior) no longer gives a warning.
- `deparse()` now deparses complex vectors in a way that re-parses to the original values. (PR#15534, patch based on code submitted by Alex Bertram.)
- In some extreme cases (more than 10^{15}) integer inputs to `dpqrxxx()` functions might have been rounded up by one (with a warning about being non-integer). (PR#15624)
- Plotting symbol `pch = 14` had the triangle upside down on some devices (typically screen devices). The triangle is supposed to be point up. (Reported by Bill Venables.)
- `getSrcref()` did not work on method definitions if `rematchDefinition()` had been used.
- `KalmanForecast(fast = FALSE)` reported a (harmless) stack imbalance.
- The count of observations used by `KalmanRun()` did not take missing values into account.
- In locales where the abbreviated name of one month is a partial match for the full name of a later one, the `%B` format in `strptime()` could fail. An example was French on OS X, where 'juin' is abbreviated to 'jui' and partially matches juillet. Similarly for weekday names.
- `pbeta(x, a, b, log.p = TRUE)` sometimes underflowed to zero for very small and very differently sized `a, b`. (PR#15641)
- `approx()` and `approxfun()` now handle infinite values with the "constant" method. (PR#15655)
- `stripchart()` again respects reversed limits in `xlim` and `ylim`. (PR#15664)